

**Microsoft® Developer
Seminar-in-a-Box Series**

**Visual Basic 4.0
Demonstration Script**

Introduction

This demonstration script was developed for the “*Building Solutions with Microsoft Visual Tools*” Seminar. Files mentioned in this document can be found on the CD-ROM distributed with the seminar materials.

It is essential that the person doing the demonstration be familiar with Visual Basic 4.0 Enterprise Edition, Microsoft SourceSafe, and the development disciplines associated with creating Client/Server databases.

Before doing this demonstration, be sure to read the associated Whitepapers located on the seminar CD-ROM .

Setup Instructions

This demonstration assumes that you have at least **two machines** running in a networked environment (preferably using TCP/IP). In the two machine environment assumed, you should have the ‘Server’ machine running Windows NT 3.51 Server or Workstation with SQL Server 6.0 installed and running. Furthermore, the Client machine assumes that you are running Windows NT 3.51 Workstation or Windows 95 with Visual Basic 4.0 Enterprise Edition. In theory you could do the demonstration with only one Windows NT machine, but you lose the effectiveness of the Remote OLE Automation demonstration (people just will not understand the concept). Regardless, this demonstration script assumes only two machines which we will refer to as the *Server* and the *Client*.

As always, you should rigorously test your demonstration multiple times before presenting to an audience. In particular, ensure that the Remote Automation Connection Manager and the Automation Manager are functioning properly and that the correct OLE Classes are registered and visible in the Remote Automation Connection Manager.

Server Machine System Requirements

Specification	Minimum Requirements
CPU	486 DX2/33MHz
RAM	24 M
Free Disk Space (for the demo files)	5 M (after NT and SQL Server)
Operating System	Windows NT 3.51 (Workstation or Server)
Additional Software	<ul style="list-style-type: none"> • SQL Server 6.0 • Visual Basic Remote Automation Connection Manager and Automation Manager

Client Machine System Requirements

Specification	Minimum Requirements
CPU	486 DX2/33MHz
RAM	16 M
Free Disk Space (for the demo files)	5 M
Operating System	Windows NT 3.51 Workstation or Windows 95
Additional Software	<ul style="list-style-type: none"> • Visual Basic 4.0 Enterprise Edition with (Remote Automation Components and OLE Controls) • Visual SourceSafe 4.0 • ODBC Driver 2.5 or higher • Microsoft Excel for Windows 95

Client Machine Visual Basic Demonstration Setup

Create a directory called "C:\VBDEV" on the client machine. Copy all of the files from the corresponding directories on the CD-ROM, under ..\Seminar\Demo\Files\VBDev. All sample applications that demonstrate database design will access one of two files: a Microsoft Access database or a Microsoft SQL 6.0 database.

The Microsoft Access database is located in the \VBDEV\DATABASE directory and is called INVOICE.MDB. All examples that open this database use the Visual Basic class INI.CLS located in the \VBDEV\SHELL directory. When you execute one of these examples, the code in the CLS.INI file tries to determine the location of the MDB by looking in the Window's Registry under:

/HKEY_CURRENT_USER/Software/VB And VBA Program Settings/VBDeveloper/Access

If it fails to locate this Registry entry, it will display a dialog box asking the user to enter the correct path which points to this database file.

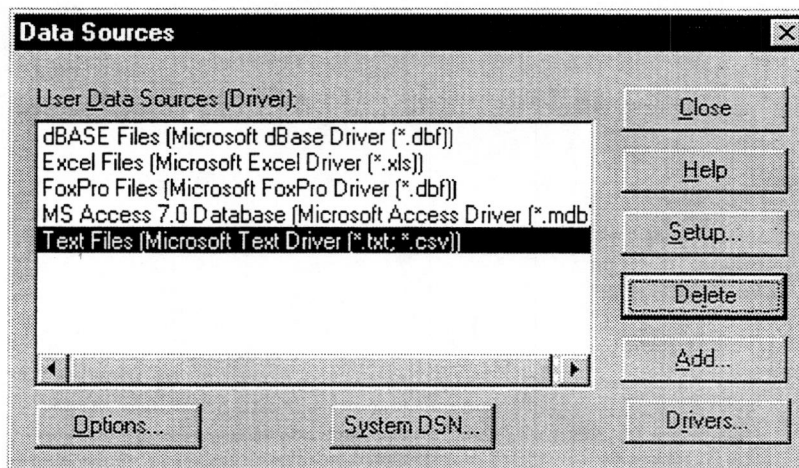
Warning: The default path that the dialog box uses is typically *not* the correct path. If you copied the files straight off the CD, the INVOICE.MDB is located in the \VBDEVDatabase directory. By default, the dialog box uses the current path of the application. You will need to change the path to point to the proper directory. If you enter the path incorrectly, you will need to use REGEDT32.EXE to change the settings manually because the dialog will not appear again unless the Registry setting is manually removed.

After entering the correct path into the dialog box, all examples, from that point forward, will be able to open the Microsoft Access database, INVOICE.MDB.

When preparing for the Microsoft SQL demonstration, you need to make sure that the ODBC driver is properly configured. Check your Control Panel for the following ODBC icon:



If it does not exist, you can install a copy from the Visual Basic 4.0 Enterprise Edition CD. After opening the ODBC Manager, you should see the following dialog:



Select the Add... button, select SQL Server for the “Installed ODBC Drivers”, press OK, and fill in the following information (except for “Server:” in which you would enter the Server running Microsoft SQL Server 6.0):

Note: If you are going to use one machine for the presentation and have installed Microsoft SQL Server 6.0 locally, enter **(local)** for the “Server:” setting.

ODBC SQL Server Setup

Data Source Name: SQL60

Description: Microsoft SQL Server

Server: MyServer

Network Address: [Default]

Network Library: [Default]

Options >>

OK

Cancel

Help

Login

Database Name: INVOICE

Language Name: [Default]

☒ Generate Stored Procedure for Prepared Statement

Translation

☐ Convert OEM to ANSI characters

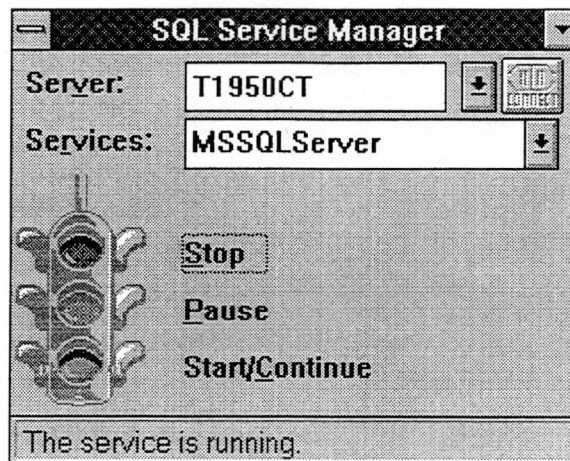
Select...



Server Machine Visual Basic Demonstration Setup

On the Server machine, make a network connection to the \VBDEV directory of the Client machine. Use a standard installation when installing Microsoft SQL Server 6.0 and be sure to install it into the following directory, C:\SQL60.

After installing SQL Server, create a table called INVOICE by executing the following steps:

1. Open the SQL Service Manager
2. Under "Server:" should be listed you server name and the "Services:" should be set to "MSSQLServer"
3. Make sure that the traffic signal has the green light illuminated (See below). If it doesn't, either double-click on the green light or press CTRL-C followed by ENTER.



4. Open Microsoft SQL Enterprise Manager
5. Select the option Query Analyzer... under the Tools menu
6. From the toolbar, select the "Load SQL Script" tool button which looks like 
7. Traverse into the directory /VBDEV/Database/Scripts
8. Load the query "INVDB.SQL" first
9. To execute this query, press the "Execute" tool button which looks like 
10. Follow steps 8 and 9 but with the files "INVTBL.SQL" followed by "INVDATA.SQL"

Note: If you installed Microsoft SQL 6.0 for the first time, you may need to register the server before executing the above steps. To register the server, follow these steps:

1. Open Microsoft SQL Enterprise Manager
2. Under the Server menu, select Register Server...
3. Enter the Server's name. If the server is running on the same machine as you are running the Enterprise manager, enter the network name of the machine you are using.
4. Make sure you use "sa" as the Login ID and DO NOT use a password.
5. Press the Register button and you are done.

Setting up the SourceSafe Database (Client only)

To correctly demonstrate the features of Visual SourceSafe, you will need to install SourceSafe 4.0 from the Visual Basic 4.0 Enterprise Edition CD. When installing, be sure to install the server as well as the client files onto the Client machine. If you are already using Visual SourceSafe on the Client machine for your own work, you should first back up the entire SourceSafe\DATA directory for safety reasons.

Add the \VBDEV\EXAMPLES\EXAMPLE.VBP project to the Visual SourceSafe Code Management Library.

Visual Basic 4.0

Feature	Narrative	Keystrokes
Visual Basic IDE	<p>For those of you who are new to Visual Basic, let me start by showing you the basics of the Visual Basic IDE (Integrated Design Environment). For those of you who have some experience with Visual Basic, I'll also point out some of the enhancements we've made in this version. First, when you start a new Visual Basic project, you begin with a blank form and controls with which the end-user interacts. Since Visual Basic is a <i>visual</i>, event-driven development tool, projects are typically made up of a collection of forms. You also see a few other windows; let me cover those first:</p> <ul style="list-style-type: none"> • The Menu/Toolbar window floats at the top of the project for easy access from anywhere. • The Toolbox contains all of the controls which you may want to place on your form. • The Project window shows all of the different files which make up your project like Forms, Class modules, or Code Module files. <p>One great new feature is the ability to keep any of these windows on top of the form on which you're working. I often want to develop apps which contain a maximized form, so I'll maximize this. But when I want to place controls on it, I used to have to restore it to get to my Toolbox. Now, I can just right-mouse click on the Toolbox and tell it to always remain on top. Now, even if I have my form maximized, I still have full access to all of my controls. To place a control on a form, I just select it, then drag it out in the location on the form where I want it.</p> <p>The next step in building a form is to set properties for a control. I do this with the Properties window. Here I have access to modify all of the properties for the form itself, as well as the controls.</p>	<p>If there are any other applications open close or minimize them.</p> <p>Launch Visual Basic.</p> <p>Point out the Menu/Toolbar window</p> <p>Point out the Toolbox window.</p> <p>Point out the Project window.</p> <p>Maximize Form1.</p> <p>Restore Form1 and click on the Toolbox.</p> <p>Right-mouse click on the Toolbox and select Always on top from the pop-up menu.</p> <p>Maximize Form1.</p> <p>Select the Command button control and drag it out in the middle of the form.</p> <p>Right-mouse click on the Command button and select Properties.</p> <p>Right-mouse click on the Properties window and select Always on top.</p> <p>In the Properties window, set the Caption property to "VB 4.0".</p>

Feature	Narrative	Keystrokes
	<p>Next, I need to respond to users by writing code. I do this by writing the code 'behind' the control itself. When I double-click on a button I get the code window with the default event or procedure for that control. You see here that I can also code for any of the events for this button. Each control supports a different set of events, depending on what purpose it serves. We'll just tell Visual Basic to pop up a message box when I click on this button.</p> <p>For those of you who like printing out your code, and hated having to go to each procedure and hit Print, you'll be glad to know that you can now display and print all of the code for a module or form simultaneously by changing a setting under the Editor Options.</p> <p>In Visual Basic you can test your application as you develop it. Just hit the Run button and you're in debug mode. First we'll place a breakpoint on this line I just wrote.</p> <p>With the addition of VBA, some nice new debugging tools are added as well. When I hit my break point, I can just right-mouse click on a variable or return value and add it to my Instant Watch window. As my program continues, and the variable is evaluated, I can see what it is.</p>	<p>Double-click on the Command button to bring up the code window.</p> <p>Click on the Proc combo box and scroll through the events. For the Click event, type in this line:</p> <pre>iRet = MsgBox("Visual Basic 4.0 is Great!")</pre> <p>Restore Form1</p> <p>From the Tools menu, select Options. Select the Editor tab and check the Full Module View and Procedure Separator boxes. Click the OK button.</p> <p>In the Code windows, select another procedure to show the two together.</p> <p>Select the line of code you wrote. Hit the F9 key. Click the Start button on the toolbar. When the app runs, click the command button.</p> <p>In debug mode, right-mouse click the iRet in the line of code. From the pop-up menu, select Instant Watch... then click Add. Click on the Debug window to bring it to the front, then point out the variable watch. Hit the F8 key to step through the code, and point out the value for the variable in the Watch window. Click the End button on the toolbar.</p>

Feature	Narrative	Keystrokes
Using the Tab Dialog Control	<p>Let me start by opening a new project and this time demonstrate the tab dialog control. I must point out that there are two tab controls: SSTAB and TABSTRIP. The primary difference is that TABSTRIP is 32-bit only whereas SSTAB has both a 16- and 32-bit OCX file. In this example I will use the SSTAB.</p> <p>Notice that I get three different tabs here.</p> <p>Notice if I select the tab and click the right mouse button that I invoke a context-sensitive menu.</p> <p>I will now select the Properties menu. Notice the Tab Caption and how I can change it. Also, please notice that the tabs are zero based.</p> <p>I will now enter a caption for all my tabs.</p> <p>Notice that I can change any of the properties listed in this dialog box. I can TabCount, TabsPerRow, TabHeight, Etc.</p> <p>I can now click on any tab and add controls to that tab. For this example, I will add two text boxes and one command button.</p> <p>I will now go to the next tab, notice that none of the previous controls are here. I will now add two more text boxes and another CommandButton. On the third tab, I will add two text boxes. But lets say that I wanted the CommandButton to appear on all three tabs. To add a control that is not owned by the Tab Control, I can either de-select the tab control or double click on the toolbar for the CommandButton. I will now place the CommandButton in the lower right corner and double-click it. Within the Click event, I will add the following code.</p>	<p>If there are any other applications open close or minimize them.</p> <p>Click the File New button to create a new project.</p> <p>Click the SSTAB control and drag the control so that it fits approximately the entire form.</p> <p>Click each tab to show how the tabs are fully operational.</p> <p>Right-Mouse Click on the SSTAB control.</p> <p>Select the Properties menu option. Change the TabCaption to read "Customer."</p> <p>Use the arrow buttons to select the next tab and enter "Order" for the TabCaption. For tab 3, enter "Order Description."</p> <p>Click on the Orientation and Style properties to let the students view the contents of the drop-down ComboBox.</p> <p>Add two TextBoxes and one CommandButton to the Tab #1 (where is not important).</p> <p>Repeat above on Tab #2.</p> <p>Repeat above for Tab #3 except leave out the CommandButton.</p> <p>Double-Click the CommandButton on the ToolBar. This will place the button in the center of the form. For clarity, drag the button to the lower right corner of the tab control. Double-click it and add the code "UNLOAD ME" and close the code window.</p>

Feature	Narrative	Keystrokes
	<p>Notice that if I click the three tabs, the CommandButton follows them.</p> <p>Now I will execute this application. Notice that the tabs still operate the same and that the third CommandButton stays in the same place on each tab. I will now click this button—watch as the application closes down.</p> <p>So that's the Tab Dialog Control.</p>	<p>Click each tab to show to show that the last CommandButton follows the tabs. Execute the application. Show that the tabs still work, and click the third CommandButton to exit the application.</p>
<p>New Custom Controls (Tab Control)</p> <p>(Data-Bound ListBox)</p>	<p>When you install your sample code disk, you will have a number of sample applications. Lets take a look at the EXAMPLE.VBP. Lets execute this example.</p> <p>Notice that a Tab Dialog control example is included in the EXAMPLE project.</p> <p>The next example I will show you is the data-bound ListBox. This control is bound to a Microsoft Access database. Now watch what happens when I click on any element in the ListBox. You will see that all the text boxes are updated with the corresponding information based on the selected record. You will also see that I have four buttons to add, delete, save, and close. Notice that the save is grayed out and will not become enabled unless I edit one of the text boxes. I will go into more detail later in this presentation when we discuss the five methodologies in access databases.</p> <p>Now I am going to stop the application from running.</p> <p>This is the control we used to place this data-aware ListBox on the form. This control is supported by both the 16-bit and 32-bit editions of Visual Basic.</p>	<p>Open the file EXAMPLE.VBP under the \VBDEV\EXAMPLES directory.</p> <p>Execute this project.</p> <p>Click on the button labeled "Tab" and briefly demonstrate the tab control and close it.</p> <p>Click on the button labeled "List Box".</p> <p>Click on some of the elements in the ListBox to show how the text boxes update.</p> <p>End the application by clicking on the Stop button in the VBIDE.</p> <p>Show the students the data-aware ListBox control by placing the mouse cursor over the button on the toolbar</p>

Feature	Narrative	Keystrokes
	<p>Let us look at the properties of the ListBox. If you'll notice, the RowSource property is set to datCust which is the name of our data control. If we show the properties of the data control, under DatabaseName we will see a path to the Microsoft Access database. If we then look at the RecordSource property, we will see how the table was defined for this data control. A table can be defined as a table name, a Microsoft Access SQL Select statement, or a Microsoft Access Query - in this case, it is an SQL Select statement.</p> <p>If you want the best performance, the Microsoft Access Query is the best choice. This is because the SQL engine is slowed down by the processing of the SQL statement which it must do each time the form is loaded and the table is inefficient because it tries to load all fields from the table when we may only need a few. The Access Query has the advantage of already being processed, can filter records, and has access to all indexes in the table.</p> <p>Now lets look at one of the TextBoxes. If we look at the properties, we see that the property DataSource points to our data control and the property DataField points to the field that we want displayed in the text box.</p> <p>To make this applet work, we need to add code to the click event of the ListBox in order to have all the TextBoxes update with the new record information. Let's look at the ListBox click event.</p> <p>This one line of code assigns the bookmark of the database the current selected item in the ListBox. This causes the record pointer in the data control to be updated to this new record. This in turn causes all the data-bound text boxes to update their information as well.</p> <p>Note that using a data-bound ListBox is more efficient than a regular ListBox because it does its own buffering of the records. Normally, with a regular ListBox, you would need to either load <i>all</i> records into the ListBox (very time consuming) or write your own code to load only parts of the data at one time.</p>	<p>Using the project window, open the file v_ex2.frm. Click on the ListBox and show the properties of it (hotkey is F4). Show the property RowSource.</p> <p>Then select the data control and show the DatabaseName property and RecordSource property.</p> <p>Click on any of the text boxes and show the values in the DataSource property and DataField property.</p> <p>Double-Click on the ListBox control.</p>

Feature	Narrative	Keystrokes
(Data-Bound ComboBox)	<p>The next example shows a data-bound ComboBox.</p> <p>First you will notice that there is an additional data control, <code>datStates</code>. If we look at its properties, you will notice that the property <code>DatabaseName</code> is pointing to the same database and the property <code>RecordSource</code> is pointing to the table called <code>tblStates</code>.</p> <p>If we look now at the data-bound ComboBox, we have the property <code>RowSource</code> as data control <code>datStates</code> and the <code>ListField</code> as the field <code>sState_cd</code>. You will also see two additional properties: property <code>DataSource</code> is set to data control <code>datCust</code> and <code>DataField</code> is set to field <code>sState_cd</code>.</p> <p>If we run the application, you will see that as we change the records using the data-bound <code>Listbox</code>, the data-bound ComboBox is updated as well. If we drop down the ComboBox, you will see the data that is loaded from the <code>datStates</code> data control using the field <code>sState_cd</code>. If we change the value in the ComboBox, this will cause a change in the field <code>sState_cd</code> associated with the data control <code>datCust</code>.</p> <p>As an option, you could also use the <code>datStates</code> data control to fill the data aware ComboBox but use DAO as the <code>DataSource</code>. We will talk later in this presentation about the advantages of this technique</p>	<p>Close the current form and open the form <code>v_ex3.frm</code>.</p> <p>Click on the data control <code>datStates</code> and show its properties. Show specifically the DatabaseName and RecordSource properties.</p> <p>Click on the data-bound ComboBox and show its properties.</p> <p>Show specifically the RowSource, ListField, DataSource, and DataField properties.</p> <p>Execute the project and select the button labeled "ComboBox."</p> <p>Show how the ComboBox is automatically updating by jumping to different records. Also drop the ComboBox down to show that it is field with a table of state abbreviations.</p> <p>Close this form and stop the application.</p>
(Grid Control)	<p>We will now take a look at the grid control. If we look at this next example we will see a grid control and a data control. If we look at the data control's properties, again we see a <code>DatabaseName</code> and a <code>RecordSource</code>. If we jump to the properties of the Grid, we see a <code>DataSource</code> property tying it to the data control.</p> <p>If I now click on the Grid control and select properties from the context-sensitive menu, you will see a properties dialog where I can manipulate the properties of the Grid control.</p>	<p>Open the form <code>v_ex4.frm</code>.</p> <p>Click on the data control and show the RecordSource and DatabaseName properties.</p> <p>Click on the Grid control and show the DataSource property.</p> <p>Right-Click on the grid control and select "Properties" from the menu.</p>

Feature	Narrative	Keystrokes
(Status Bar)	<p><i>[As an instructor, take time to familiarize yourself with the properties of the grid control and take time in class to explain what many of the properties do]</i></p> <p>Now lets run this application again and see this control in action. As you can see, I can select and edit any of the records in this Grid but I cannot add or delete records because of the properties set for the Grid.</p> <p>I will now stop this application, modify the properties to allow the addition and deletion of records and will re-run the application and demonstrate.</p> <p>That concludes the Grid control.</p> <p>These next two controls we will look at are only available under the 32-bit version of Visual Basic. The first control is called the Status Bar control.</p> <p>Here is an example of the status bar. As you can see, it is identical to the status bars you typically see on most Windows 95 applications. You will also notice that it too has many properties that we can use to modify its appearance.</p> <p>Let execute this applet and see how this example uses the status bar. Notice that the status initially contains the text "Company Name". This was done to indicate that the field with the current focus is prompting the user for a company name. As we tab around the text boxes, notice that the status bar is constantly updated. You will also notice that when I take my mouse pointer and move it over a particular text box, the status bar updates to show me which text box my mouse is currently above.</p>	<p><i>[Custom Demonstration]</i></p> <p>Execute the project and select the button labeled "Grid".</p> <p>Edit a grid cell in the Sales Amount Column to show that editing is possible.</p> <p>Stop the application, go to the properties of the Grid control, change the properties AllowDelete and AllowAddNew to True.</p> <p>Execute the project and select the button labeled "Grid".</p> <p>Use the down arrow key to move past the last record, enter a new record. Move off that record to save it. Now demonstrate the delete capabilities by deleting that newly added record.</p> <p>Open the form v_ex5.frm from the project window. Use your mouse to select the status bar so that students can identify with it. You might also point out the button from the toolbar that was used to create the status bar.</p> <p>Execute the project and click the button labeled "Status Bar".</p> <p>Use the TAB key to move focus to each text box.</p> <p>Move the mouse cursor over each of the TextBoxes to demonstrate the status bar picking up mouse movement.</p>


Feature	Narrative	Keystrokes
(RichTextBox Control)	<p>Let us return to the code of this control. First, let's look at the property dialog window for this control. I can invoke it by right-mouse clicking on the control and selecting properties. Let's take a look at some of the properties that are available to us.</p> <p><i>[As an instructor, take time to familiarize yourself with the properties of the status bar control and take time in class to explain what some of the properties do. It might be helpful to demonstrate how to add another panel to the status bar. Be sure to point out that the status bar panels are 1 (one)- based, not 0 (zero)- based like the Tab control.]</i></p> <p>If we take a quick look at the code, we can see that the sub txtCompany_GotFocus() makes a call to the sub StatusMsgShow(). You will also see that the sub txtCompany_MouseMove() has the one line in it as well. Lets take a look at the StatusMsgShow() subroutine.</p> <p>The first line here checks to see if the status message we are sending is already being displayed in the text box. If not, the status bar's panel is updated to the new text.</p> <p><i>[If time available, you could demonstrate the hotkeys SHIFT-F2 and CTRL-SHIFT-F2 during the above example]</i></p>	<p>End the application.</p> <p>Right-Mouse click on the StatusBar control and select "Properties".</p> <p><i>[Custom Demonstration]</i></p> <p>Double-click on TextBox labeled txtCompany. This should bring you to the GotFocus event. Now jump to the MouseMove event.</p> <p>Now jump to the subroutine StatusMsgShow().</p>
	<p>Let us now look at the other 32-bit control that we will be demonstrating - it is the RichTextBox control. You will see that we have, what looks like, a regular multi-line text box. The difference is in the amount of control we have over the text. This TextBox supports the RTF (Rich Text Format) text formatting standard. As you can see, I can highlight some words, press CTRL-I (or italics) or CTRL-B (for bold). I can even change the font mid-way through the text.</p> <p>Looking at the buttons below, you will see that you have the ability to write and read RTF formatted files.</p>	<p>Execute the project and select the button labeled "Rich Text Box."</p> <p>Highlight some text within the RichTextBox and press CTRL-I and CTRL- B. Also, drop down the Edit menu to show other options this example has available.</p>

Feature	Narrative	Keystrokes
	<p>Let us take a look at the code behind this. To give you a better idea as to the number of properties available to you, I will invoke the Visual Basic help system and look under "RichTextBox" control and click on the jump, "Properties".</p> <p>As you can see, we have the ability to change color, font, alignment, indenting, underlining, italics, bolding, and much more.</p> <p>Lets take a look at the code behind the "Save" button. As you can see, it uses the CommonDialog control to create a standard Windows "Save As ..." dialog box. Saving the actual text was done in one line. This line calls the RichTextBox's SaveFile method and passes to it the filename and the file type. Valid file types are RTF and plain ASCII Text.</p>	<p>End application and Click on the form v_ex6.frm from the project menu.</p> <p>Set focus to the RichTextBox and press F1. Under "Index" Tab, search for "RichTextBox Control." Double-click invokes the help text for the RichTextBox control. Now jump to the help on "Properties" by clicking the word. Scroll down to the properties that begin with "SEL."</p> <p>Double click on the Save button and show students the line starting: rtbNotes.SaveFile...</p>
Using Classes	<p>Let take a look at an example that uses classes to manipulate strings. Currently, this string class can only perform two tasks: it can strip characters from a string and parse a DOS Path into its components. But for demonstration purposes, this will give a good look at how a class works.</p> <p>First, let us look at the click event for the "StringStrip" button. The first line of the function creates a string object, oStr, based on the string class, clsString. As you know, an object can contain one or more properties as well as one or more methods. In the next line, we assign a value to the property, "TheString". The final line calls the method, "StringStrip", passes a SPACE value, and takes the result and passes it to the MessageBox Function.</p>	<p>If you do not have this project loaded already, load the project EXAMPLES.VBP.</p> <p>Click on the form V_EX7.FRM from the Project menu.</p> <p>Double-click the button labeled "StringStrip" in order to bring up the code window which should default to the click event. Highlight the first line starting "DIM...."</p> <p>Highlight the line starting "oStr..."</p> <p>Highlight the line starting "MsgBox..."</p>

Feature	Narrative	Keystrokes
	<p>Let's now take a look at the code behind the class. Here you see two variables defined, <code>pstrString</code> and <code>pstrConvString</code>. Both variables are defined as <code>Private</code>. This indicates that no procedure outside of this class can access these variables. This is a good example of encapsulation. So, you ask yourself, "how can I pass information to these variables?" That can be done using the new Visual Basic 4.0 function types called the "Property Let" and "Property Get". If we look at the "Property Let <code>TheString</code>" function, we can see how the property "<code>TheString</code>" was created. All this function does is take the value passed to it and assigns it to the <code>pstrString</code> variable. If we want the "<code>TheString</code>" property to be write-only, we would only have a Property Let function but in this case, we also want to allow our users to read the property as well, so we include a Property Get function. All this function does is assigns the value currently in <code>pstrString</code> to the function name "<code>TheString</code>".</p> <p>Now let us look at the method "<code>StringStrip</code>". As you can see, methods are simply public function inside the class. Any function defined as <code>Private</code> will only be accessible by other functions within your class.</p> <p>This function simply strips out all characters based on the character passed to it and places the return value into the private variable, "<code>pstrConvString</code>". We then assign the variable to the return value of the function which in turn passes the value to the caller of the method. Any questions?</p> <p>Now let's look at the click event for the "File Commands" button. In this next block of code, you will see that the first step we perform is assigning the property "<code>TheString</code>" to the value of text that is in the TextBox labeled "<code>txtFile</code>". The next four steps are calling four different methods of our string class and copying the values to four other textboxes.</p>	<p>Double-Click on the <code>v_string.cls</code> file in the project listbox. You should have the General Declarations showing at this time.</p> <p>Jump to the Property Let function "<code>TheString</code>".</p> <p>Jump to the Property Get function "<code>TheString</code>".</p> <p>Jump to the function "<code>StringStrip</code>".</p> <p>Open the <code>v_ex7.frm</code> form again and view the click method for the "File Commands" button.</p>

Feature	Narrative	Keystrokes
	<i>[From here, you, as the instructor, can show the associated code in the string class and reinforce the ideas of designing Properties and Methods.]</i>	<i>[Custom Demonstration]</i>
Add-Ins	<p>Let us first look at the code inside our sample Visual Basic Add-In.</p> <p>As you can see, there is no default form associated with this project. Our Add-In is actually an OLE Server. To define this as an OLE Server, we need to open the Project Options Dialog Box. As you can see, we have selected the radio button for the StartMode as OLE Server. We have also defined the StartupForm as "Sub Main". This will indicate which function gets executed first when the project is first started. You will also note that the ProjectName is called "CommandAdd." Let us now look at the code in "Sub Main".</p> <p>All Visual Basic Add-Ins, in order for Visual Basic to load them, must be listed in the VB.INI file under the subheading "Add-Ins32" (or "Add-Ins16" for 16-bit Visual Basic). As we look at our Sub Main, we see that its only purpose is to verify that there is a listing in the VB.INI for the Add-In called "CommandAdd.AddCommand." As you can see, Sub Main is the only routine in the ADDIN.BAS file.</p> <p>Now let's look at the class itself. For an OLE Server to link properly with the Visual Basic IDE, three procedures must exist: ConnectAddIn(), DisconnectAddIn(), and AfterClick(). There are also three standard object variables that should be declared: an environment object, VBIDE.Application; a menu object, VBIDE.MenuItems; and a menu line object, VBIDE.MenuLine.</p> <p>Lets take a look at the ConnectAddIn() procedure. In this routine we obtain a handle to the current instance of the Visual Basic Environment (<i>this is important because Visual Basic 4.0 allows multiple copies of the environment to be running</i>), a handle to the Add-Ins menu, and we create a new menu item called "Command Button."</p>	<p>Open the Project ADDIN.VBP.</p> <p>From the menu option Tools, select Options. Them select the Project Tab.</p> <p>Close the Project Options dialog and go the Sub Main for the Add-Ins project.</p> <p>Close the ADDIN.BAS module.</p> <p>Open the ADDIN.CLS module.</p> <p>Show the declarations for the ADDIN class module.</p> <p>Open the ConnectAddIn() subroutine.</p>

Feature	Narrative	Keystrokes
	<p>The last line in the procedure passes the Add-In object itself to the Menu Object. This is a good example of polymorphism. All the menu object is required to do when a user clicks it is call your object's "AfterClick" method. The Visual Basic environment doesn't need to know what your method does or how it does it, it just knows that if it calls that method, the object will know what it needs to do.</p> <p>If we look at the AfterClick event, the first thing it does is define a TemplateControl. This is a generic object that can reference any control on a form. The following line references the TemplateControls collection (which is a collection of all controls on a form) and adds to the collection another CommandButton. The only function this Add-In performs is to add a Command Button to the current form.</p> <p>Finally, if we look at the DisconnectAddin() event, we can see the clean code that removes our Add-In from the Add-In menu list.</p> <p>Now lets see our Add-In in action. First, we must execute this project. Now that the OLE Server is running, we need to open another instance of Visual Basic 4.0. (Normally, you would compile your Add-In into an EXE and not need two instances of Visual Basic running, but for demonstration purposes, this may help in your understanding of how an OLE Server works). If we go into the Add-Ins Manager, we can see our registered custom Add-In, "CommandAdd.AddCommand". Next we need to select it in order to add it to our Add-Ins menu bar.</p> <p>If we bring down the Add-Ins menu again, we can see that a new menu option is added called, "Command Button". If we select our new menu option, you will see that it added a CommandButton control to the current form.</p> <p>That is how Add-Ins are done.</p>	<p>Open the AfterClick event.</p> <p>Execute the Project.</p> <p>Minimize the current Visual Basic Environment and execute another instance of Visual Basic 4.0.</p> <p>Open the Add-Ins menu and select "Add-In Manager".</p> <p>Select the check-box of the line: "CommandAdd.AddCommand" and press the button labeled "OK".</p> <p>Open the Add-Ins menu and select "Command Button".</p> <p>Close and Exit both instances of Visual Basic.</p>

Feature	Narrative	Keystrokes
Visual SourceSafe	<p>When I open the EXAMPLES project in Visual Basic, you'll notice that Visual SourceSafe has prompted me for a password. This is because I've added this project to the SourceSafe code management library.</p> <p>I'll just type in my user ID and password. When I do so, SourceSafe loads the project. As we look at the different files which make up this project, we notice that there are un-checked check boxes next to each of the files. When I open one of the module files and try to edit it, something interesting happens—I'm unable to edit it. This is due to the fact that, while I have the file loaded into Visual Basic, I do not have it checked out from SourceSafe, so I cannot edit it.</p> <p>Just by right-mouse clicking on a file in the project window, I can check files in or out. Right now, I want to check all of the files out. Keep in mind that I can still run and 'test' a project without checking the files out, but if I want to make modifications, then I have to check the file(s) out.</p> <p>Now notice that the icons next to the files have changed indicating that the files are checked out.</p> <p>Now when I go to edit a file, it lets me do so.</p> <p>Let's take a close look at what other advantages Visual SourceSafe provides me.</p>	<p>Locate and open the Example.vbp file.</p> <p>Open any module and try to type anything in. Point out the Error message, then dismiss it. Close the module.</p> <p>Right-mouse click on any file in the project window and select Check-out from the shortcut menu. Point out the changed check boxes in the Project window.</p> <p>Double-click on the any module again and at the top of the code windows, type in: 'This is my Visual Basic 4.0 app. (don't forget the apostrophe at the beginning of the line).</p>
	<p>From the Add-Ins... menu, I can go directly to the SourceSafe Explorer. Here I can view and manage all of the different projects I have currently under source code control. You see here, that I do, in fact, have the Examples project as well as other projects. I could also have sub-projects for each if I wanted.</p> <p>In the right hand pane, I have all of the files which make up the project.</p> <p>If I've just done some work which brings this project up to a specific development milestone, I can add a label to the project.</p>	<p>From the Add-Ins menu, select SourceSafe and then Run SourceSafe.</p> <p>In VSS, point out the different projects.</p> <p>Point out all of the files in the Examples project.</p> <p>In the left pane, click the Examples folder, then click the Label button</p> <p>on the toolbar . Type in appropriate information in to the fields and click OK.</p>

Feature	Narrative	Keystrokes
<p>Data Control</p>	<p>These next five demos will look identical. The only difference between them is how they access their respective databases. By the end of this section, you should have a good enough understanding to be able to select one method over another based on your project's requirements.</p> <p>First, we will look at the Data control.</p> <p>As you can see, we have a Data control, <code>datCust</code>, with properties <code>DatabaseName</code> and <code>RecordSource</code> set to define where the data is coming from.</p> <p>If we look at all the TextBoxes, we can see that each one is bound to our Data control, <code>datCust</code>, through the property <code>DataSource</code>.</p> <p>Let's take a look at the code behind the "New" button. The first line sets a module level variable to true. This variable acts as a flag to let other procedures know that we are in edit mode. The next line calls the Data control's "AddNew" method. This tells the Data control to clear its record buffer.</p> <p>The next line calls a local procedure "DataHasChanged" that enables and disables certain buttons in order to prevent the user from damaging the database.</p> <p>The final line sets focus to the first text box. Because the user will most likely enter data after clicking the "New" button, this makes your application more user friendly.</p> <p>Now let's look at the Delete procedure. We first dimension a local variable. We then create a beep noise and prompt the user with a message asking if they are sure that they want to delete the current user. It is good practice to always prompt the user before irreversibly deleting any information. If the user selects "Yes," we then call the Data control's Delete method and Requery method.</p>	<p>Open the project <code>CUSTDC.VBP</code>. View Form <code>CUST.FRM</code>. Select the Data control and press F4 to view control properties. Point out the DatabaseName property and RecordSource property.</p> <p>Select some of the TextBoxes on our Cust form and point out in the properties window the DataSource property.</p> <p>Double-click the button labeled "New" on the form Cust which will invoke a code window showing the <code>cmdNew_Click</code> procedure.</p> <p>Move the blinking cursor in the code window somewhere in the "DataHasChanged" text and press SHIFT-F2.</p> <p>Press CTRL-SHIFT-F2 to return to the <code>cmdNew_Click</code> procedure.</p> <p>Double-click on the button labeled "Delete" on the form Cust.</p>

Feature	Narrative	Keystrokes
	<p>The Requery method causes an update to occur on the Data control to ensure that we are not still showing the now deleted record in our ListBox.</p> <p>Notice in our IF...THEN statement that we used the constant "vbYes" to indicate a "Yes" response. This is a constant defined in the Visual Basic for Application (VBA) library. To see a list of all constants, open the Object Browser.</p> <p>Under the <u>L</u>ibraries/Projects drop-down ComboBox, select "VBA".</p> <p>We then select "Constants" from the <u>C</u>lasses/Modules ListBox. As you can see, in the <u>M</u>ethods/Properties ListBox, there are many predefined constants available for you to use in any of your Visual Basic 4.0 application. Any questions?</p> <p>Now, let's look at the "Save" button. This has only two lines. The first line calls the Data control's "Update" method. This physically writes the data in the record buffer to the disk drive. The next line calls a local procedure that resets all the buttons to the state they were in when we first executed this form.</p> <p>The last button is the "Close" button. As you remember from the "DataHasChanged" procedure, if the "New" button is clicked, this button's caption is labeled "Cancel". Therefore, the first line in this procedure checks to see if the button was clicked to close or to cancel. If it was clicked to close, then unload the form. Otherwise, perform a cancellation. When we cancel, we need to determine if we are canceling while we are in "Edit Mode" or "Adding Mode". This is where that flag, "mboolAdding", is used. We check this flag to see which mode we are in.</p>	<p>Press F2 (the hotkey for the Object Browser).</p> <p>Select "VBA - Visual Basic for Applications" in the <u>L</u>ibraries/Projects ComboBox.</p> <p>Select "Constants" from the <u>C</u>lasses/Modules ListBox.</p> <p>Close Object Browser.</p> <p>Double-click the button labeled "Save" on the Cust Form.</p> <p>Double-click the button labeled "Close" on the Cust Form.</p>

Feature	Narrative	Keystrokes
	<p>If we are in "Adding Mode," we call the Data control's "CancelUpdate" method which erases all data in the record buffer and re-loads the record it had in it prior to the "AddNew" method.</p> <p>If we are in "Edit Mode," we call the Data control's "UpdateControls" method which performs a re-read of the current record. This re-read causes any changes that may have occurred in the text boxes to be overwritten by the original record.</p> <p>Let's look at how we get into "Edit Mode." I will now execute the application. I will be running against a Microsoft Access database, so I click the button "Access". I now have my standard Customer Form. Watch as I edit one of the TextBoxes.</p> <p>Notice that as soon as I press any key, the buttons labeled "New" and "Delete" are disabled, the "Save" button becomes enabled, and the "Close" button is labeled "Cancel." What event do you suppose we used to switch into "Edit Mode"?</p> <p>We could have used the "Change" event of each TextBox, but in this case it was more efficient to use the "KeyDown" event of the Form. Let's take a look at the code.</p> <p>The first line checks to see if the "Save" button is disabled, because if it were enabled, we would already be in "Edit Mode". Next we check to see if the active control is the ListBox. This will prevent us from going into "Edit Mode" if the user presses a key while the ListBox has focus. Then the remaining line calls the procedure "DataHasChanged" which, as we saw earlier, puts us in "Edit Mode".</p> <p>Notice that we did not explicitly change the variable "mBoolAdding" to False. This is because we already defined it as False in our "ButtonReset" procedure and doing so again would be redundant.</p>	<p>Close Code Window.</p> <p>Execute the application.</p> <p>Press the button labeled "Access MDB".</p> <p>Set focus to the TextBox with the label "First Name".</p> <p>Press any lettered key.</p> <p>Press the button labeled "Cancel".</p> <p>Press the same button which is now labeled "Close".</p> <p>Press "ALT-F4" to close the final window.</p> <p>Double-click anywhere on the form that does not contain a control. This should invoke the code for the "Form_KeyDown" procedure.</p>

Feature	Narrative	Keystrokes
	<p>(Remember, "mBoolAdding" is the flag we check to determine if we are in "Edit" mode or "Adding Mode".)</p> <p>Another important point to mention here is that the "KeyPressed" event of the form requires the form's "KeyPreview" property to be set to true in order for it to trap key strokes.</p> <p>Let's go back to the "ButtonReset()" procedure. As you remember, this procedure toggles most of the CommandButtons to be disabled. Let me show you another, more efficient way to accomplish this.</p> <p>In this procedure, instead of manually toggling each control, I set up a loop that goes through every control in the form's control collection to check to see if it is a CommandButton. If it is, I toggle the control's "Enabled" property. The benefit of this procedure is that it is generic and can be called from any form—plus this code is actually faster.</p> <p>[Optional: If you, the instructor, wish to demonstrate this function, follow the instructions to the right.]</p> <p>The last thing we are going to look at is the code behind the original form that comes up that is used to select which data source we are going to use.</p> <p>As you can see by the code, all we are doing is setting our frmCust's public property, "pboolSQL60", to False.</p> <p>Then, in the other button, it sets that same property to True.</p> <p>Now, let's go to the "Form_Load" event of the form frmCust. The first line creates a new object based on our INI class. Let's say that this class is responsible for handling our data source information. We check the form's "pboolSQL60" variable and call a method based on it.</p>	<p>Open the procedure "ButtonReset" located in "General" section of the form's code module.</p> <p>Open the procedure "ButtonToggle" which is also located in the "General" section.</p> <p>[Optional: Go to the procedures "cmdClose_Click" and "cmdSave_Click" and comment out the line "Call ButtonReset" and uncomment the line "Call ButtonToggle" and re-execute.]</p> <p>End the application if it is running.</p> <p>Open the MAIN.FRM Form.</p> <p>Double-click the "Access MDB" button.</p> <p>Open the "cmdSQL_Click" event.</p> <p>Open the FRMCUST.FRM form.</p> <p>Open the "Form_Load" Event.</p>


Feature	Narrative	Keystrokes
	<p>We then execute the method "INIGet" that looks in our Registry (or INI file for a 16-bit application) and loads the appropriate connection information into other object properties. The last two lines query those properties to get the necessary information for the Form's "Connect" and "DatabaseName" property.</p> <p><i>[The importance of showing the audience this procedure is to point out that the Data control can run off of different data sources without changing many lines of code.]</i></p> <p><i>[Optional: If you have time, you may go into the INI Class and show some of its code.]</i></p> <p><i>[Optional: If you have time, you may re-execute the application and press the SQL Server button to demonstrate that this code indeed works.]</i></p> <p>That is the Data control. We will now take a look at the Data Access Object and see how it differs from the Data control.</p>	<p>Close the code window. Close the project.</p>
Data Access Objects	<p>Let's take a look at an example that uses the Data Access Object.</p> <p>I would like to start by looking at the code in the "Form_Load" event.</p> <p>The first line calls a function that sets two module level variables. One of them is a flag to indicate whether we are in "Adding" mode or "Edit" mode. The other is a flag indicating whether or not the recordset has any data in it.</p> <p>The next line in our "Form_Load" event procedure calls a module level function called "JetOpen".</p>	<p>Open project CUSTDAO.VBP.</p> <p>Open the "Form_Load" event for the frmCust Form.</p> <p>Place the cursor on the word "FormInit" and press SHIFT-F2. You should now be in the "FormInit" procedure.</p> <p>Press CTRL-SHIFT-F2. You should now be back in the "Form_Load" procedure.</p>

Feature	Narrative	Keystrokes
	<p>Looking at the code, you can see we define an object based on our familiar INI class. Again we have the IF...THEN line that checks to see if we are running against an Access database or a SQL Server database. Based on that flag, we call the method that processes our database selection and creates our public database object, "gdb". We created the database object by calling the "OpenDatabase" method of the DBEngine object. The last line sets the return value of the function to True.</p> <p>Back in the "Form_Load" procedure, you can see that if a True is returned, we call two functions. The first, "cboStateLoad", loads the state data into our cboState ComboBox. The second procedure loads our Customer data into our Customer ListBox. Let us look at the "cboStateLoad" procedure first.</p> <p>For those of you that used Visual Basic 3.0 and the DAO object, this code should look different than what you are used to. First we dimension a Recordset object. Then we assign an SQL statement to a local string variable. We then call our database object's "OpenRecordset" method which creates a snapshot based on our SQL string that was passed to it. The next few lines is a standard loop that traverses through each record of our Snapshot and adds it to the State ComboBox. The last two lines close the snapshot and removes the recordset object from memory. This last line is important, because unless you remove the object from memory, it will stay there for the life of your application.</p> <p>[Optional: You can point out to the students that the "!" (pronounced "Bang") is used when the text to the right of the object does not represent a method or a procedure. In this case, it represents a user defined field name.]</p>	<p>Place the cursor on the word "JetOpen" and press SHIFT-F2. You should now be in the "JetOpen" procedure.</p> <p>Return to the "Form_Load" procedure.</p> <p>Jump to the cboStateLoad procedure.</p>

Feature	Narrative	Keystrokes
	<p>Now let's quickly jump to the <code>lstCustLoad</code> procedure, the second procedure that is loaded prior to the form being loaded.</p> <p>The first few lines declare a local string variable and clear the <code>ListBox</code>. The next set of lines checks to see if the form level variable "<code>mdsData</code>" is pointed to a valid recordset or not. At the time when the form is initially being loaded, obviously it would be set to "Nothing". The next line defines our SQL statement, and then the next line calls the "<code>OpenRecordset</code>" method of our database object and assigns the resulting recordset to the variable "<code>mdsData</code>". Notice that we are opening a dynaset this time instead of a snapshot. The primary difference is that a dynaset is editable whereas a snapshot is not. Snapshots have the advantage of loading faster than dynasets if your recordset is less than 500 records.</p> <p>The next set of lines is a basic DO UNTIL loop that loads each record of the recordset into the <code>ListBox</code>. Notice the line in the middle of this loop. It assigns the field "<code>lCust_id</code>" to the "<code>ItemData</code>" property of the <code>ListBox</code>. The "<code>ItemData</code>" property of a <code>ListBox</code> allows us to store a numeric value along with each Text item. Then, when a company is selected from our <code>ListBox</code>, we can obtain the "<code>lCust_id</code>" value, which provides us with a much faster search value than simply the company name.</p> <p>The last IF...THEN statement in our procedure checks to see if any elements exist in our <code>ListBox</code>. If so, we trigger a "Click" event on the <code>ListBox</code>. In a minute, I will show you why.</p> <p>Let's open the <code>ListBox</code>'s "Click" event. The first line checks to make sure that an item is selected in the <code>ListBox</code>. Then it assigns the current "<code>ListIndex</code>" to the "<code>AbsolutePosition</code>" property of our database object. Think of the "<code>AbsolutePosition</code>" property as a simple ordinal count of the records in our database.</p>	<p>Open the <code>lstCusLoad</code> procedure from the <code>frmCust</code> Form.</p> <p>Open the <code>lstCust_Click</code> procedure from the <code>frmCust</code> Form.</p>

Feature	Narrative	Keystrokes
	<p>Absolute position zero will always be the first record in our recordset. Absolute position one will always be the second record and so on. This line will ensure that our record pointer is pointing to the same record as the one that is currently selected in our ListBox. Remember, since we are not working with data-bound TextBoxes, we need to manually assign each of TextBoxes with the appropriate data. The next line does just that. Let's jump to the FormShow procedure.</p> <p>What this procedure does is first checks to see if our record pointer is pointing to a valid record and then fills each TextBox with the data in the corresponding field out of our database object. For the State field, you will notice that it has to update the ComboBox to show the proper State listing. The final line is our standard "ButtonReset" procedure that initializes our form's buttons.</p> <p>Now let's execute our project.</p> <p>As you can see, we can go against Microsoft Access or Microsoft SQL Server. For this case, we will choose Microsoft Access.</p> <p>As you can see, if we select some of the items in our ListBox, all our TextBoxes update properly. Now let's look at the code behind each of our CommandButtons.</p> <p>First we will look at the code behind the "New" Button. Here you will see our same flag, "mboolAdding", is set to True. We then call a procedure called "FormClear". Because the TextBoxes are not data-aware, we need to manually clear each TextBox. This is what the procedure "FormClear" does. The next line in the "cmdNew_Click" procedure calls the function "DataHaschanged" which is identical to the one shown in the previous example. Finally, the last line sets focus to the Company TextBox.</p>	<p>Open the FormShow procedure of the frmCust Form.</p> <p>Close any code windows.</p> <p>Execute Project.</p> <p>Click the button labeled "Access MDB".</p> <p>Click on five or more companies listed in the ListBox.</p> <p>Click the "Close" button and press ALT-F4 to end application.</p> <p>Open the Form Cust.FRM and double-click the "New" CommandButton. This should display the code for the "cmdNew_Click" procedure.</p>


Feature	Narrative	Keystrokes
	<p>Now we will look at the Delete button. The code here is identical to the code in the Data control example except for the line that actually does the deleting. In this example, we issue the Delete method directly from the database object instead of calling it from the Data control.</p> <p>The Save routine is a bit different from the code in our Data control. The first IF...THEN line checks to see if we are in "Adding Mode" and if so, it calls the function TableIDGet() which returns the next available ID for the table name that is passed in. <i>[If you, as an instructor, wish to explain the TableIDGet procedure, you may; but for this script, we will continue on].</i> The next IF...THEN line checks the mode flag again, and either calls the "AddNew" method or the "Edit" method directly from our database object. After that, we must go through each and every TextBox and copy the data to our record buffer. After all fields are entered, we issue the "Update" method of the database object. This physically writes the data into the datafile.</p> <p>After writing the data, the next IF...THEN line checks to see if we are in "Adding Mode" and, if so, moves the record pointer to the last record. An important fact to remember is that after the "Update" method is called, the record pointer still remains on the record it was on PRIOR to when the "AddNew" method was called. In this example, since we always increment our key index value by one each time, we are guaranteed that the record just entered will be the last record in the recordset—therefore, we call the "MoveLast" method to move the record pointer to the last record.</p> <p>The remaining two lines call the "lstCustLoad" procedure which updates our TextBoxes and "ButtonReset" which updates our buttons.</p>	<p>Open the cmdDelete_Click procedure in the frmCust form.</p> <p>Open the FormSave procedure in the frmCust form.</p>

Feature	Narrative	Keystrokes
	<p>Now let's look at the "Close" button. Remember that the procedure "cmdClose_Click" supports both the "Close" and "Cancel" buttons. Like the previous example, we check to see what state the button is in. If we are closing, then we unload the form. If we are canceling, then we need to re-load the previous record back into the TextBoxes. As you see in the code, we make a call to the "FormShow" procedure which loads the TextBoxes based on the currently selected ListBox item, followed by a call to the procedure "ButtonReset" which resets our buttons back to their original states. The last thing we check is if there are any records in our database object; we do this by checking to see if the "EOF" property is True. If it is, then we trigger the "Form_Activate" Event.</p> <p>This procedure checks the "mboolAddRecords" variable which is a flag that indicates whether records exist in the database object. As if you remember, back in the Form_Load event we called the procedure FormInit which sets this variable to True if no records existed in our current database object. If no records exist, we prompt the user with a MessageBox that tells them there are no records and asks if they would like to start entering some. If the user replies "yes", we invoke the "Click" event of the "New" button. Otherwise, we unload the form.</p> <p>Any questions before we leave the Data Access Object?</p>	<p>Open the "cmdClose_Click" procedure from the frmCust form.</p> <p>Open the "Form_Activate" procedure from the frmCust form.</p> <p>Close the application.</p>
ODBC Setup	<p>[This demo does not require Visual Basic.]</p> <p>Let's take a look at what was involved in setting up our Microsoft SQL Server link using the ODBC Administrator program. First, I will go into my Control Panel.</p> <p>Then, will execute my ODBC Administrator application.</p>	<p>Execute the Control Panel located in the "Main" Group box under Windows NT or using the Start Button under "Settings" for Windows 95.</p> <p>Execute the icon labeled "32bit ODBC".</p>  <p>ODBC</p>

Feature	Narrative	Keystrokes
	<p>The ListBox labeled "User Data Sources (Driver)" contains all the current ODBC data sources we have defined on our system. Let's suppose we want to add a data source. I would first click the "Add..." button.</p> <p>Now I have a list of all available ODBC drivers installed on the system. Let's select the "SQL Server".</p> <p>Here I need to create a unique DSN (Data Source Name) that I can use to access my data via ODBC, in this case, I will choose to call it "SQL60".</p> <p>The only other thing I need to define is the server. So, under "Server:", I will select my Windows NT machine.</p> <p>That's it.</p>	<p>Click the "Add..." button.</p> <p>Select the item "SQL Server" from the ListBox and click the "OK" button.</p> <p>Type in "SQL60" into the TextBox next to the "Data Source Name:" label.</p> <p>Under the label "Server:", select the network name of the machine which has SQL Server loaded. If SQL Server is located on your local machine, type "(local)".</p> <p>Click the "OK" button. Click the "Close" button. Close the Control Panel.</p>
ODBC API	<p>Let's look at the ODBC API example application.</p> <p>Here we have the "Form_Load" event; it is fairly similar to our DAO example. We first call the "FormInit" function which sets some module level variables. The line calls the function "ODBCInit". As you will see in this example, all functions beginning with "ODBC" are wrapper functions for the actual ODBC API function. As we move through this code, you will see the API function calls as we go.</p> <p>In the "ODBCInit" function, you see that the first function we call is "ODBCEnvInit". This function is a wrapper for the API function "SQLAllocEnv" which checks to see if the ODBC driver is loaded and available. The next function is "ODBCConnectInit", that is a wrapper for the API function "SQLAllocConnect", which established a memory area where we can put login information that will be used to login to our data source. That concludes the "ODBCInit" function.</p>	<p>Open the project CUSTOAPI.VBP.</p> <p>Open the Form_Load procedure of the frmCust form.</p> <p>Open the ODBCInit procedure in the modODBCAPI code module.</p>

Feature	Narrative	Keystrokes
	<p>The next line in the "Form_Load" event is a call to "ODBCDataOpen".</p> <p>In this function, we first get the necessary information out of the INI class, just like we did in the previous examples, and then we call the function "SQLDriverConnect". This function will go out and establish a connection with our ODBC driver. We then check to see if the connection was successful; if it is, we return True.</p> <p>Back in the "Form_Load" event, we assumed that the ODBC initialization worked and we loaded the State ComboBox and the Customer ListBox, like we did in the DAO example. Lets look at the "cboStateLoad" procedure.</p> <p>In this procedure, we define our SQL statement and call the function "ODBCQuerySubmit", which is another wrapper function for the ODBC API functions.</p> <p>As you can see in the function "ODBCQuerySubmit", we call the ODBC API function "SQLAllocStmt" which allocates memory for the SQL statement. We execute the SQL statement by calling the ODBC API function "SQLExecDirect".</p> <p>Now that our SQL statement has been executed, our record pointer should be pointing to the first record in our back-end database. We then read the first record into our record buffer by executing the ODBC API function "SQLFetch()". Now that the first row of data is in memory, we execute the procedure "ODBCData". This procedure is a wrapper for the ODBC API function "SQLGetData" which returns the data based on the column number passed to it. In this case, "STATE_CODE", which is a user-defined constant that holds the integer value of the column for the State abbreviations, is passed to "SQLGetData". After obtaining the first state abbreviation, we add it to our State ComboBox and call "SQLFetch" again to load the next record. This continues until all records are loaded.</p>	<p>Open the "Form_Load" procedure in the frmCust form.</p> <p>Open the "ODBCDataOpen" function in the modODBCAPI module.</p> <p>Open the "Form_Load" procedure in the frmCust form.</p> <p>Open the "cboStateLoad" procedure in the frmCust form.</p> <p>Open the "ODBCQuerySubmit" function in the modODBCAPI module.</p> <p>Open the "cboStateLoad" procedure in the frmCust form.</p>

Feature	Narrative	Keystrokes
	<p>The final line in this procedure is the ODBC API function "SQLFreeStmt", which frees the memory used by the SQL statement.</p> <p>Now that we have loaded our State ComboBox, we need to load our Customer ListBox. The code is very similar to the code we used to load the State ComboBox. We first define the SQL statement. Then we call the "ODBCQuerySubmit" function which submits the SQL statement. We then enter a loop that calls the "SQLFetch" and the "ODBCData" function until all records are loaded into the Customer ListBox. In the last line, we free up the memory used by the SQL statement and invoke the "Click" event of the ListBox.</p> <p>Does anyone know what we need to do before we can call this ODBC API functions?</p> <p>The answer is, "declare them." As you recall from our section on calling the Windows API, we need to declare any API function that we want to use. Let's go to the ODBC code module and quickly glance at all the declare statements required.</p> <p>You can see that conditional compiling statements are required because the API calls for 32-bit operating systems are different from those of 16-bit operating systems.</p> <p>Now let's take a look at the ListBox's "Click" event to see how the TextBoxes are populated with data. The "lstCust_Click" procedure is identical to the other examples, but let's look specifically at the "FormShow" procedure.</p> <p>The first few lines declare some constants and dimension some variables. Then, we create a complex SQL statement and call the "ODBCQuerySubmit" function, which creates our recordset. Then, we use the "SQLFetch" function to load the first record into the record buffer.</p>	<p>Open the "lstCustLoad" procedure in the frmCust form.</p> <p>Open the ODBCAPI.BAS code module.</p> <p>Open the general declarations section of the ODBC API module. Scroll slowly through this entire function showing the students the amount of up-front coding that is involved.</p> <p>Close the code window.</p> <p>Open the lstCust_Click procedure in the frmCust form.</p> <p>Open the FormShow procedure in the frmCust form.</p>

Feature	Narrative	Keystrokes
	<p>The next ten or so lines are calls to "ODBCData" that must load in each column of the record separately and then copy it to the TextBox. Finally, the last few lines free up the record buffer memory.</p> <p>As you can see, more coding is involved for even the simplest of tasks.</p> <p><i>[The demo could stop at this point. The audience should be aware of the amount of programming involved when using the ODBC API. If the instructor has additional time, he/she may look at some of the code behind the "Delete" and "New" buttons.]</i></p>	
Remote Data Control	<p>Let's open our customer database application again, only this time we will use the Remote Data control.</p> <p>Notice that the icon is different from our regular data control. The Remote Data control icon has a little satellite on it. Remind the students that this icon only comes with the Enterprise edition and it only appears in the 32-bit Visual Basic Environment.</p> <p>As you can see from the properties list, connecting to our Microsoft SQL Server is as simple as filling in properties. Notice the "DataSourceName" property is set to SQL60 which, if you remember, is the DSN name we gave when we were using the ODBC Administrator. The property "SQL" contains our SQL statement, which in this case is loading all records in order of company name.</p> <p>This is our data-bound ListBox. This ListBox is no different from the ListBox that was used for the Data control example. Our data-bound TextBoxes are also the same.</p> <p>As you can see, we have the property "DataSource" set to rdcCust and the "DataField" property set to szCompany_nm.</p>	<p>Open the project, CUSTRDC.VBP.</p> <p>Move the mouse cursor over the following icon in the Visual Basic Toolbar:</p>  <p>Open the CUST.FRM form. Click on the Remote Data control and press F4 to get the property list.</p> <p>Close properties window.</p> <p>Select the ListBox "dblCust".</p> <p>Select the TextBox to the right of the label called "Company:"</p> <p>Press F4 to show the properties window.</p>

Feature	Narrative	Keystrokes
	<p>Let's now look in the "Form_Initialize" event. Those who have worked in Visual Basic 3.0 may remember that it used to be that the "Form_Load" was the first event to be triggered when a form was first loaded. In Visual Basic 4.0, forms are now true objects and all objects, whether they be forms or user-defined objects, have constructor and destructor methods. Constructors and destructors are used in all object-oriented languages and will always be the first and the last methods executed, respectively. The syntax for the constructor and destructor methods in Visual Basic 4.0 is "{object}_Initialize" and "{object}_Terminate".</p> <p>As you can see in our "Form_Initialize" event, we create an INI object that we use to call the registry and obtain the information we need to set the DSN, UserName, and Password. This is all the code we need to start our application. This ListBox and all the TextBoxes will load automatically. Now the only events we need to worry about are the "New", "Edit", and "Delete" events.</p> <p>Let's look at the procedure behind the "Click" event of the "New" button. This looks very similar to the regular Data control. We set our "Adding" flag to True, call the "AddNew" method of the Remote Data control, call the "DataHasChanged" procedure to update our buttons, and set focus to the first updateable field.</p> <p>If you look at our "Delete" button, you can see we do the same thing we did earlier. We prompt the user to make sure that they intend to permanently delete the current record, call the "Delete" method, and call the "Refresh" method.</p> <p>For the "Save" button, we check to see if we are in "Adding" mode or "Edit" mode and, based on the mode, we either call the "Update" method or the "UpdateRow" method of the Remote Data control.</p>	<p>Open the "Form_Initialize" procedure in the frmCust form.</p> <p>Open the "cmdNew_Click" procedure in the frmCust form.</p> <p>Open the "cmdDelete_Click" procedure in the frmCust form.</p> <p>Open the "cmdSave_Click" procedure in the frmCust form.</p>

Feature	Narrative	Keystrokes
	<p>Finally, on our "Close" button, we check to see if the caption is "&Close" or "&Cancel". If it is "&Close", we unload the form. Otherwise, we check to see if we are in "Adding" or "Edit" mode and either call the "CancelUpdate" method or "UpdateControls" method. This procedure is nearly identical to the one in the Data control example. As you can see, converting an application that uses the Data control to one that uses the Remote Data control is very simple.</p>	<p>Open the "cmdClose_Click" procedure in the frmCust form.</p> <p>Close project.</p>
Remote Data Object	<p>Let's take a look at using the Remote Data Object with our customer database application. First, I would like to step into the Form_Load event.</p> <p>As you can see, it is similar to our DAO example; we first call "FormInit" which, if you remember, initializes our two flags. Then we call "RDOOpen".</p> <p>Here you can see we create an INI object and, again, use it to obtain the information necessary to open our database. We then call the rdoEngine object's "OpenConnection" method, which, if you remember from the DAO example, is similar to calling the "OpenDatabase" method of the DBEngine object, and assign the result to our database object variable, "grdoConn".</p> <p>Back in our Form_Load procedure, we see that the remaining lines call our "cboStateLoad" and "lstCustLoad" procedures.</p> <p>In "cboStateLoad", you can see that instead of creating a Recordset object, we create an rdoResultset object, build our SQL statement, and then call the database object's "OpenResultset" method. We then loop through all the records and add them to our ListBox—just like we did in the DAO example.</p>	<p>Open project CUSTRDO.VBP.</p> <p>Open form frmCust and open procedure Form_Load.</p> <p>Open procedure RDOOpen from the frmCust form.</p> <p>Open procedure Form_Load from the frmCust form.</p> <p>Open procedure cboStateLoad from the frmCust form.</p>

Feature	Narrative	Keystrokes
	<p>If we look in our "lstCustLoad" procedure, we find the same steps as we did in the DAO example. We clear the ListBox, make sure that our resultset object is not already set, create our SQL statement, and then call the "OpenResultset" method. Finally, we traverse through all records in the result set, add them into our ListBox, and fire the ListBox's "Click" event.</p> <p>Now lets look at the "Click" event. Again, just like in the DAO example, we assign the "ListIndex" property of the ListBox to the "AbsolutePosition" property of our database object. Then we call the "FormShow" procedure.</p> <p>In the "FormShow" procedure, we check to see if any records exist in the table. If so, we copy all the fields in the record into the TextBoxes.</p> <p><i>[From here it should be apparent that programming with Remote Data Object is very similar to the Data Access Object. If you would like to continue, you can go into the code of the "New", "Delete", and "Save" to reinforce this similarity.]</i></p>	<p>Open procedure lstCustLoad from the frmCust form.</p> <p>Open procedure lstCust_Click from the frmCust form.</p> <p>Open procedure FormShow from the frmCust form.</p> <p>Close project.</p>
Application Framework	<p>Let's now take a look at two applications that were developed using the same application framework. Both applications are order entry application; one going against a Microsoft Access database and the other against Microsoft SQL Server 6.0.</p> <p>Lets look at the application that is running against Microsoft Access. As you can see in our project window, we have 23 modules: 8 general modules, 8 class modules, and 7 form modules. As you will see in a minute, this application was designed so that if we wanted a different back end, we could swap out the 8 classes with different classes.</p>	<p>Open project CUST.VBP from the JET directory.</p> <p>Scroll through the project ListBox to show the audience all the files.</p>

Feature	Narrative	Keystrokes
	<p>Let's first take a look at the Sub Main. This is in your workbook under the topic, "Sub Main()". It is good practice to start all your applications with a Sub Main because it allows you to set up the environment better prior to showing the first form. It also helps to have a standard Sub Main that all your application share.</p> <p>In this Sub Main procedure, you see we start by changing the mouse cursor to an hourglass. We display copyright information and follow that by calling our login class. If the login is successful, we call the "DataInit" method and the "DataOpen" method. Notice that so far in our code, all function calls have been kept generic. Based on this code, you cannot tell which back end you are running against. Furthermore, the back end could be changed and this code will still operate the same.</p> <p>Let's take a look in our "DataOpen" method. In this example, you can see that the "DataOpen" method checks to see if our database object, "odb", is defined. Based on the result, it calls our INI class to get the database location information and issues the method, "OpenDatabase" on our database object.</p> <p>Let's now return to your Sub Main procedure. I am now going to load another instance of Microsoft Visual Basic 4.0 and load into it the Customer project located in the \ODBC directory.</p> <p>Let's take a look at the two project lists side by side. What do you notice here? They are almost identical. In actuality, all the modules that have the same name in both projects are the same files.</p>	<p>Open the procedure "Main" out of the modAppCode.BAS file.</p> <p>Move the cursor somewhere within the text "DataOpen" and press SHIFT-F2.</p> <p>Press CTRL-SHIFT-F2.</p> <p>Close all code and form windows. Minimize this instance of Visual Basic and load another instance of Visual Basic and load the project CUST.VBP from the \ODBC directory.</p> <p>Position the project window of one Visual Basic instance next to the project window of the other Visual Basic instance.</p>

Feature	Narrative	Keystrokes
	<p>The files that begin with "J_" are files specific to the Microsoft Jet Engine and the files that begin with "O_" are files specific to Microsoft SQL Server via ODBC.</p> <p>Lets go to the Sub Main of the ODBC project. As you can see, this is the same code. Now, let's look at the DataOpen method.</p> <p>This time we are now referencing the ODBC class. Notice that we are calling functions like "SQLConnect". Let's look in the workbook under the topic, "DataInit()". Listed here are the two "DataInit" functions from the Jet class and the ODBC class. Notice that they accomplish the same basic task but in two different ways. But remember, our Sub Main doesn't care how it performs the DataInit task, it just knows that when it calls it, the class will take care of it. Any questions about what we have covered so far?</p> <p>Remember, the rule here is "no data access in your forms". The minute you put direct data access within a form, that form is restricted to one and only one data access methodology.</p> <p>Let's take a look at this rule in action. I am now going to open the form "frmCust" from our Jet project and look at the "Form_Load" function. As you can see, we call our standard procedures, "FormInit", "cboStateLoad", and "lstCustLoad". Let's open "FormInit".</p> <p>Notice something new here, Visual Basic cannot determine which "FormInit" we wish to jump to. Because we stuck to our application framework, each of our forms has a method which uses the same name. This, again, is another great example of polymorphism. For now, I am going to press cancel and I now want to direct your attention to the "cboStateLoad" procedure.</p>	<p>Minimize the instance of Visual Basic with the "J_" classes in them.</p> <p>Open the Main procedure in the APPCODE.BAS module.</p> <p>Move the cursor somewhere within the text "DataOpen" and press SHIFT-F2.</p> <p>Minimize this instance of Visual Basic</p> <p>Maximize the instance of Visual Basic with the "J_" class modules. Open the form "frmCust". Open the procedure "Form_Load".</p> <p>Move the cursor somewhere within the text "FormInit" and press SHIFT-F2.</p> <p>At the dialog box, click on the button labeled, "Cancel".</p>

Feature	Narrative	Keystrokes
	<p>Notice that this code is still inside the form, "frmCust". Therefore, we are still restricted from making direct data access calls. We do this by creating an object, oState, based on our state class, clsState. We then call the methods, "GetFirstRow" and "GetNextRow" of the state class in order to move the record pointer and read the property "pstrStateCode" in order to get at the table's data. Remember, this is not direct data access. We can still redefine the state class to go against another type of back-end data source.</p> <p>Let's jump to the "GetFirstRow" method.</p> <p>Notice that we have the "GetFirstRow" method defined in our customer class (clsCust), our invoice class (clsInvoice), our invoice items class (clsInvItem), our produce class (clsProduct), our user class (clsUser), and our state class (clsState).</p> <p>Why? So that the user of the class only needs to learn the method "GetFirstRow" once. The user doesn't need to worry about whether they are getting the first row of customers, products, or whatever. They know that when they call the "GetFirstRow" method, they get the first row.</p> <p>Let's jump into the customer class. Because we are in the customer class of our Jet classes, you see we have the line which creates a snapshot by calling our database object's "OpenRecordset" method.</p> <p>Let's now return to the "cboStateLoad" function. Remember that we are back in our form module for the frmCust form. I am now going to minimize this instance and go to the instance of Visual Basic that contains the ODBC classes.</p>	<p>Move the cursor somewhere within the text "cboStateLoad" and press SHIFT-F2.</p> <p>Move the cursor somewhere within the text "GetFirstRow" and press SHIFT-F2.</p> <p>Click the clsCust item in the ListBox and click "OK".</p> <p>Press "CTRL-SHIFT-F2"</p> <p>Minimize this instance of Visual Basic and open the instance which contains the "O_" classes.</p>

Feature	Narrative	Keystrokes
	<p>Here we are in the "FormInit" of the ODBC project. Again, we create an object based on our state class, we call methods "GetFirstRow" and "GetNextRow", and retrieve data using the property "pstrStateCode". This is the same code we were using before.</p> <p>Now let's jump into the "GetFirstRow" method. As you would expect, instead of calling the "OpenDatabase" method, we call the "ODBCQuerySubmit". Let's now jump to the "GetNextRow" method.</p> <p>Again, we have our "SQLFetch" procedure which references our ODBC API.</p> <p>Any questions?</p> <p>As a lead into our next section, what we are going to do is physically remove the data access classes into their own project and create an OLE server out of them. That way our customer form will now reference the object within an OLE server instead of the object within its own EXE.</p>	<p>Open form "CUST.FRM". Open procedure "FormInit".</p> <p>Open the "GetFirstRow" method of the Customer class.</p> <p>Move the cursor somewhere within the text "GetNextRow" and press SHIFT-F2. Select the frmCust item and click "OK".</p> <p>Close all code and form windows.</p>
Excel Calculator	<p>Let's take a look at an example of a Visual Basic application using Excel to add two numbers together. This example will demonstrate how to move data from Visual Basic to Microsoft Excel and back again.</p> <p>First we will open the example "ExcelCal". In order to access Microsoft Excel, we need to make sure that we have the reference accessible by our Visual Basic application. To do this, we go into our Reference Dialog Box and make sure that we have the "Microsoft Excel 5.0 Object Library" checked.</p> <p>Now lets take a look at this example. I think the best way to demonstrate this is to simple step through it. First, I will place a breakpoint in my CommandButton's "Click" method.</p>	<p>Open the project "OLESERV\ExcelCal.VBP"</p> <p>From the Visual Basic environment open the Reference Dialog Box by selecting the "Rferences" option from the "Tools" menu.</p> <p>Open the procedure "Cmd1_Click" from the "Form1" form.</p>

Feature	Narrative	Keystrokes
	<p>Now I will run the application.</p> <p>I will enter two numbers.</p> <p>Here we are in our "Click" method. Now let's step through the code. I will do this by pressing the F8 key. This one line will take a little bit of time because it is loading Microsoft Excel into the background. We now have a reference to our Microsoft Excel Worksheet in the variable "X". We now pass the values in the two TextBoxes into cells in our Excel worksheet.</p> <p>Here we modify a third cell with an Excel macro that adds the two cells together.</p> <p>We now take the value in that cell and assign it to the third TextBox.</p> <p>We remove Microsoft Excel from memory.</p> <p>We remove our object variable from memory.</p> <p>If we go back to our form, we see that the correct value is showing.</p> <p><i>[If the instructor wishes, he/she can uncomment the line that reads "X.Application.Visible ...". This will make Microsoft Excel visible and the audience can actually watch as data moves into cells.]</i></p>	<p>Place the edit cursor somewhere in the line "Set X = ..." and press F9.</p> <p>Execute the application. Enter one number into each of the TextBoxes. Click on the button labeled "Ask Excel".</p> <p>Press F8.</p> <p>Press F8. Press F8.</p> <p>Press F8.</p> <p>Press F8.</p> <p>Press F8.</p> <p>Close the code window.</p>
OLE Server #1	<p>Here is a simple example of an OLE Client and OLE Server written in Visual Basic 4.0.</p> <p>Here you see the only lines of code in the entire program. The first line creates an instance of the class ReturnDate. Notice that we need the project name, DateReturn, as well as the class name when referencing a Visual Basic 4.0 OLE server.</p>	<p>Open the project OLETEST1.VBP.</p> <p>Open the "cmdDate_Click" procedure.</p>

Feature	Narrative	Keystrokes
	<p>In a minute, I will show you how to register this class. The next line creates a message box that displays the return string from the method "DateStarted".</p> <p>I will now open a second instance of Visual Basic 4.0 and open our server application.</p> <p>Let's look in our project windows. As you see, we have only two files: a class module and a general code module. If we look quickly at the general code module, we will see that there is a Sub Main and that's it. The only reason for including a Sub Main is because Visual Basic 4.0 requires a starting point for all Visual Basic applications.</p> <p>If you notice in the options dialog box under the "Project" tab we have our Sub Main defined as the startup form. We also defined our project name as "DateReturn". Remember? This is the name we needed to in our OLE Client project to create the instance from the class. Also within this dialog box, we have the StartMode set to "OLE Server".</p> <p>Now let's open our class.</p> <p>I will press F4 to show the properties of our class. Here you see three properties. The "Name" property is simply the name of our class. The "Instancing property" can be set to one of three settings. The first is simply no instancing. This will prevent any application from creating an instance of this class. The second is single use instancing. This means that one instance will be create for each application that requests one. For example, if ten users make requests to the OLE server, ten separate servers will be loaded into memory. The benefits of this selection are apparent when you have few requests and speed is very important.</p>	<p>Open a second instance of Visual Basic 4.0. Open the project OLESERV1.VBP.</p> <p>Display the project window.</p> <p>Open modMain</p> <p>From the Visual Basic "Tools" menu, select "Options...". Select the "Project" Tab.</p> <p>Close this dialog box by clicking the "Cancel" button.</p> <p>Open the "ReturnDate" class and show the code in the General Declarations section.</p> <p>Press F4.</p>

	<p>The last is multiple use instancing which is the most popular. This will load one OLE server to service all requests.</p> <p>The last property is "Public". This simply sets whether outside OLE clients can invoke instances of this class.</p> <p>Now, before I execute these applications, I want to set a couple breakpoints. The first I will put in the procedure "Class_Initialize".</p> <p>Now I will execute the application</p> <p>As you can see, not very exciting. Since no application is requesting instances of our OLE server, the program remains dormant.</p> <p>I will now go back to our OLETEST1 project and set a breakpoint in our "Click" method.</p> <p>Lastly, before I execute this application, I need to set the reference to my new OLE server. I will now go into my Reference Dialog box.</p> <p>Here you will find our OLE server listed as "Return a Date". Once I select it, you will notice that the "Location:" is set to our Visual Basic project. Now I am ready to go.</p> <p>We are now at our first breakpoint. You will notice that we have not instantiated our class yet. When I press F5, we should jump into our Initialize routine.</p> <p>And we do. Here we set the private variable "pdtDate" to the current date and time. When I hit F5, the program will resume and our message box should appear.</p> <p>That is all the code it takes to create an OLE server.</p>	<p>Close property window</p> <p>Open the "Class_Initialize" procedure. Place the edit cursor anywhere in the line "pdtDate" and press F9.</p> <p>Execute the application.</p> <p>Set focus to the instance of Visual Basic with OLETEST1 Open the "cmdDate_Click" procedure. Place the edit cursor anywhere in the line "MsgBox ..." and press F9.</p> <p>Under the "Tools" menu option, select "References..."</p> <p>Select the line "Return a Date" and click "OK". If any instances of "Return a Date" have 'missing' in front of them, use another instance.</p> <p>Start application. Press the button labeled "Get Date".</p> <p>Press F5.</p> <p>Press F5.</p>
--	--	--

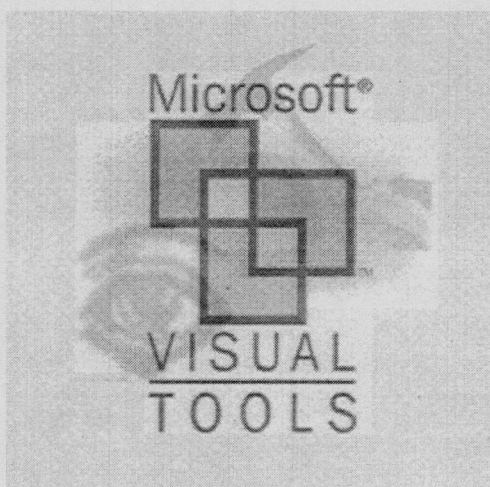
Feature	Narrative	Keystrokes
OLE Server #2	<p>This next OLE Server example is a little more complicated than our first example. Like before, I will open two instances of Visual Basic 4.0; one containing the project OLETEST2 and the other OLESERV2.</p> <p>In our example, we will show a form communicating with an OLE server, which in turn perform data access tasks. As you can see, we have two CommandButtons which will read and write data to a Microsoft Access table. Let's look at the button labeled "Read Sales" first.</p> <p>Here we create an instance of our Sales class to check to see if data exists in our Sales table. If so, we populate our three TextBoxes with the data from the first record. That's it.</p> <p>If we look at the button labeled "Write Sales", you can see we do almost the same thing. We create an instance of our sales class. Populate the three properties of our sales object with the data in our three TextBoxes. Then call the "WriteSales" method to save the data.</p> <p>In the General Declaration section of our class module, you can see our three properties defined: pstrQtr, pstrYear, and plngSales.</p> <p>In our method "GetSales", you should find this code very familiar. We have calls to our INI class to get information about our data source. We then set our database object and recordset object to point to our Sales table. Then, provided the table is not empty, we copy the data within the first record into our three properties.</p> <p>The "WriteSales" method is also very straightforward. We make calls to the INI class to get our database information. We define our database object. We then choose to use an SQL INSERT statement along with the database object's Execute method.</p>	<p>Open two instances of Visual Basic 4.0.</p> <p>Load project OLESERV2.VBP into one instance and OLETEST2.VBP in the other. Minimize OLESERV2.</p> <p>Show OLETEST2.FRM.</p> <p>Open the "cmdRead_Click" procedure.</p> <p>Open the "cmdWrite_Click" procedure.</p> <p>Minimize OLETEST2. Maximize OLESERV2. Open the code module for the QtrSales class.</p> <p>Open procedure "GetSales".</p> <p>Open procedure "WriteSales".</p>

Feature	Narrative	Keystrokes
---------	-----------	------------

	<p>Before I run this, I'd like to set a couple breakpoints first. The first is in our method "GetSales".</p> <p>Now I will execute the OLE Server.</p> <p>As you can see, it is dormant waiting to be invoked. I now go back to my OLE client.</p> <p>I would like to place a breakpoint within my "Read Sales" button.</p> <p>I now need to set my reference so that my OLE client can see my OLE server.</p> <p>Now I will execute my client.</p> <p>I will now try to read data from my sales table by clicking the button "Read Sales".</p> <p>Our breakpoint hit and we are now at the line that attempts to read data using the method "GetSales". I will hit F8 and as you will see, I jump into our OLE server.</p> <p>I am now inside the OLE server and I will step through each line as it reads the data.</p> <p>As you can see, the database was opened and data was copied into our properties. I will now hit F5 and continue on with the program.</p> <p>And now the data is in our TextBoxes.</p> <p><i>[If the instructor has more time, it might be beneficial to also step through the "WriteSales" method as well.]</i></p> <p><i>[Optional: If more time is available, you can open the OLESERV2 project and select the "Make OLE DLL File..." under the "File" menu to create a DLL.</i></p>	<p>Open procedure "GetSales". Place edit cursor in the line starting with "Set db = ..." and press F9.</p> <p>Execute project.</p> <p>Minimize OLESERV2 Maximize OLETTEST2</p> <p>Open procedure "cmdRead_Click". Place edit cursor in the line starting with "If oSales ..." and press F9.</p> <p>Under the "Tools" menu select "Refferences..." Check the line that reads "Populate Sales Table". If any instances of "populate sales table" have 'missing' in front of them, use another instance. Press "OK".</p> <p>Execute project.</p> <p>Click the "Read Sales" button.</p> <p>Press F8.</p> <p><i>[Continue to press F8 until the line "Exit Function" is the current line.]</i></p> <p>Press F5 twice.</p>
--	---	--

Feature	Narrative	Keystrokes
	<p><i>This will show the audience the speed improvement of an in-process OLE server. Remember to change the reference in the OLETTEST2 file to point to the DLL instead of the VBP.]</i></p>	
<p>OLE Server with Microsoft Excel</p>	<p>In this example I will use the same OLE server as in the previous example but instead of running a Visual Basic 4.0 OLE client, I am going to use Microsoft Excel 7.0 for Windows 95 instead. You have already seen OLESERV2 so I will go ahead and execute it.</p> <p>I will now load Microsoft Excel.</p> <p>I will now open one of my already created worksheets.</p> <p>For those who are not familiar with Excel, you can see that I have two tabs: "SalesFigures", which contains my spreadsheet data, and "Module1" which contains Visual Basic for Application (VBA) code.</p> <p>Just like in Visual Basic, we need to set our references.</p> <p>Now let's step through this module.</p> <p>The first line creates our instance of the Sales class. The next three lines set our object properties to the values within our Excel spreadsheet. The next line calls the "WriteSales" method of our Sales object and, based on the return value, either sends a message box telling the user if the data was save successfully.</p> <p>Any questions?</p>	<p>Open project OLESERV2.VBP</p> <p>Execute project. Minimize Visual Basic 4.0.</p> <p>Open Microsoft Excel.</p> <p>Open worksheet SALES.XLS .</p> <p>Press the "SalesFigure" tab.</p> <p>Press the "Module1" tab.</p> <p>Under the "Tools" menu select "References..." Check the line that reads "Populate Sales Table". Press "OK".</p> <p><i>[Press F8 throughout this dialog]</i></p> <p><i>[Note: If the data fails to save, it is most likely cause by a duplicate key violation. Before executing this code, make sure that the data in the spreadsheet does not already exist in the Sales table.]</i></p> <p>Close application.</p>

Feature	Narrative	Keystrokes
Optional: OLE Server #3	<i>[There is one more OLE server example which takes the example from the Application Framework section and splits it into an OLE Client application (OLECUST.VBP) and an OLE Server (OLESERV.VBP) application. If you have extra time, you can demonstrate this example as well.]</i>	



***Microsoft*® Developer
Seminar-in-a-Box Series**

**Visual Basic 4.0
Seminar Slides**

Legal Disclaimer and copyright slide

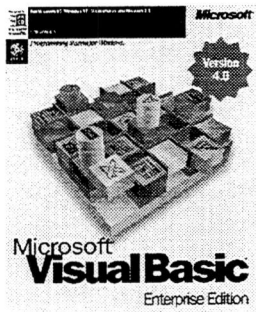
Please remove this slide prior to
presenting, but leave it in for any
electronic or material
photocopying or distribution

Microsoft, Windows, and Win32, are registered trademarks and Visual Basic is a trademark of Microsoft Corporation. All other trademarks, marked and not marked, are property of their respective owners.

This document is provided for informational purposes only. The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to change in market conditions, it should not be interpreted to be a commitment on the part of Microsoft and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

INFORMATION PROVIDED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND FREEDOM FROM INFRINGEMENT. The user assumes the entire risk as to the accuracy and the use of this document. This document may be copied and distributed subject to the following conditions: 1) All text must be copied without modification (except foreign language translation) and all pages must be included; 2) All copies must contain Microsoft's and Application Developer Training Company's copyright notice and any other notices provided therein; and 3) **This document may not be distributed within the boundaries of Canada or the United States of America.**

Copyright © 1996 Application Developers Training Company. All Rights Reserved.



Microsoft Visual Basic 4.0

Your Instructor: {Instructor Name}

Application Developers
• **Training Company** •



Microsoft's Visual Basic 4.0 Features

Developed By: Application Developers Training Company
7151 Metro Blvd., Ste 175, Edina, MN USA 55439
VOI: (612) 943-0047 FAX: (612) 942-8452

Date Created: 03/01/96

Last Update: 5/29/96 19:03

Approximate Delivery: 7.0 - 8.0 hours

Presenter:

Target Audience: Database developers looking at Visual Basic 4.0 as a new development tool.

The objectives of this presentation are:

- Provide a good understanding of where Microsoft is Positioning Visual Basic 4.0 as a developer tool
- Provide developers with a comprehensive overview of the Visual Basic 4.0 Programming Environment
- Show developers how easy it is to create Visual Basic applications
- Give developers a brief look at all the important features with Visual Basic

Main Topics

- Features in Visual Basic 4.0
- Database Development
- Application Framework
- OLE Servers

Introduce yourself and give a small description of how the day will be conducted. This presentation is broken into four sections as show above.

The first section will run approximately 3 hours and will give the students a quick look at all the important features within Visual Basic 4.0.

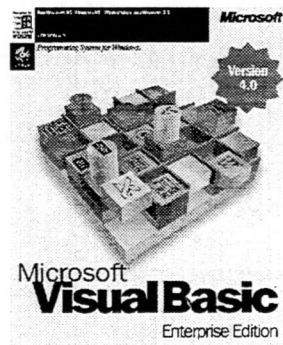
The next topic will be about 2 hours in length and will talk about the various options Visual Basic has available to access data. It will look primarily at the Microsoft Jet Engine and ODBC.

The third topic will run about 1 hour and will take a look at one possible answer as to how a developer should set up an application framework.

The last topic should run about 1 hour and will look at creating OLE servers using Visual Basic 4.0.

We recommend taking two 15 minutes breaks, one in the morning and one in the afternoon with a one hour lunch in the middle. With this schedule, this presentation should take, on average, 7.0 hours depending on how well you stick to your schedule.

Visual Basic 4.0 Features



This slide is a simple placeholder to show that we are ready to start the first section.

Seminar Objectives

- Which platforms the different flavors of Visual Basic 4.0 support?
- What are some of the features in Visual Basic 4.0?
- How do you set up a Project?
- How do you use the ActiveX™ Controls, formerly called OLE controls?

After introducing the seminar, go through each of the topics above giving an approximate 1 minute overview of each bullet point. To get you started, explain that you will be covering the three editions of Visual Basic 4.0. You will be going through many of the features in Visual Basic that appeal to application developers. You will be setting up a simple project and demonstrating many of the controls within Visual Basic. You can point out at this time that the name "ActiveX™ Controls" is simply a name change from "OLE controls".

Workbook Section: Introduction

Seminar Objectives

- What are all of the ways you can access database from Visual Basic 4.0?
- How do you use Class Modules?
- How do you create Visual Basic Add-Ins?
- What is available in Visual Basic 4.0 to help with version control?

Continuing with the overview, you will be talking about the some of the ways Visual Basic can access databases. It would be good to point out at this point that approximately 50% - 70% of this presentation will be directly related to accessing data and that you will be demonstrating using both a Microsoft Access table as well as a Microsoft SQL 6.0 table. You will also cover the latest addition to Visual Basic, the class module. You will show the audience how Visual Basic facilitates the object-oriented style of programming. Lastly, you will demonstrate using Add-Ins and how to use the version control program Microsoft SourceSafe.

Workbook Section: Introduction

Visual Basic 4.0 Platform Support

- Windows 95 (32-bit)
- Windows NT 3.51 or later (32-bit)
- Windows 95 or Windows NT 3.51 (16-bit supported with WOW)
- Windows 3.x (16-bit)
- Windows for Workgroups 3.x (16-bit)

•Explain that Microsoft's goal is to migrate Windows users to the 32-bit operating system and that Visual Basic is an excellent tool to move projects to that system. You can also add that Microsoft is also pushing Windows NT as its operating system of choice for the Client/Server environment.

•When explaining WOW (Windows On Windows), state that it gives 32-bit operating system the ability to execute 16-bit applications. You can also point out that a 32-bit application running on a 32-bit operating system will perform better than a 16-bit application running under that same 32-bit operating system.

•You can point out that you cannot use the Standard Edition of Visual Basic 4.0 to develop 16-bit applications.

•*Optional:* Point out that Visual Basic 4.0 was completely re-written from scratch. Visual Basic 3.0 was compiled using Assembly and was optimized for the 8086 architecture. Visual Basic 4.0 was compiled using Visual C++ as is optimized for the 32-bit environment. (This information should be used to support the need to move to 32-bit)

•Point out that Visual Basic 4.0 is one of only a few development languages that has this ability to generate both 16-bit and 32-bit applications. This feature is a plus for companies that are in the process of migrating machines from 16- to 32-bit operating systems.

Workbook Section: Platforms Supported

Three Exciting New Editions

- Standard Edition (32-bit)
 - Designed for the hobbyist/beginner
- Professional Edition (16 & 32-bit)
 - Designed for the individual professional developer
- Enterprise Edition (16 & 32-bit)
 - Designed for **Teams** of developers creating **Distributed, Network** applications

Briefly talk about the fact that there are three editions available to developers. The next three sides go into more detail on each of the products so don't spend too much time on each one. A good idea would be to simply point out which operating systems works with which edition and present it in such a way that you are asking the student to pick the best edition that they need. For example:

"If your project specifications require developing in both 16- and 32-bit operating system, the professional edition is the best choice"

"If your project specifications require client/server development and/or team development, the Enterprise edition might be the best choice"

Optional: You can point out that Visual Basic 3.0 and Visual Basic 4.0 should operate safely on the same development machine.

Workbook Section: Editions Of Visual Basic 4.0

Standard Edition

- Supports 32-bit development platforms such as Windows 95 and Windows NT 3.51 or later.
- This edition of Visual Basic has all of the standard Windows controls, plus data aware controls.

Talk about some of the features of the Standard Edition (slide 1 of 2):

Workbook Section: Features of the Standard Edition

Standard Edition

- No programmable Data Access Objects (DAO) are available
- Printed *Programmer's Guide*
- On-line help

Talk about some of the features of the Standard Edition (slide 2 of 2):

Workbook Section: Features of the Standard Edition

Professional Edition

- Support for both 16 and 32-bit development platforms
- All of the features found in the Standard Edition
- More ActiveX™ Controls
- Crystal Report Writer Version 3.0

Talk about some of the features of the Professional Edition
(slide 1 of 2).

Workbook Section: Features of the Professional Edition

Professional Edition

- Programmable Data Access Objects
- Ability to create OLE servers
- *Language Reference* manual
- *Professional Features* book
- *Crystal Reports* manual
- On-line help
- *Visual Basic Books Online*

Talk about some of the features of the Professional Edition
(slide 2 of 2).

If new users to OLE are present, simply say that OLE servers allow other application, such as Microsoft Word or Excel, to execute your application and exchange data between them. You can also say that this will be covered in more detail later in the presentation.

Workbook Section: Features of the Professional Edition

Enterprise Edition

- Intended for the serious corporate developer
- Documentation includes a great tutorial on client/server and ODBC connectivity.
- Information on building OLE servers and Remote OLE servers

Talk about some of the features of the Enterprise Edition (slide 1 of 3).

You may need to touch briefly on Remote OLE servers at this time. Simply state that this allows you to access OLE servers on other machines on a network. More on this will be covered later.

Workbook Section: Features of the Enterprise Edition

Enterprise Edition

- Microsoft Visual SourceSafe version control system
- Remote Data Objects (RDO) and Remote Data control (RDC)
 - Allows you to communicate through ODBC directly to Microsoft SQL Server 6.0 and Oracle 7.x

Talk about some of the features of the Enterprise Edition (slide 2 of 3).

It would be good to quickly point out the benefits involved with RDO. In the prior release of Visual Basic, the options available to developers to access Microsoft SQL or Oracle were either through the ODBC API, which had a large learning curve and was difficult to program with, or through the Microsoft Access Jet Engine which came at the expense of a lot of overhead and tended to slow the application down.

RDO (and RDC) allow you to access the ODBC drivers through an object very similar to the DAO object *except* that it doesn't go through Jet. RDO simply provides a thin layer between your code and the ODBC API. You can also point out that in some cases, RDO is even faster than making direct calls to the ODBC API because RDO contains highly optimized API calls.

Point out that RDO and RDC are only available in 32-bit.

Workbook Section: Features of the Enterprise Edition

Enterprise Edition

- With RDO you still use data aware controls
- A Component Manager allows you to manage reusable code created by several programmers
 - This should help prevent large teams from reinventing the wheel.
- Remote OLE Automation Manager

Talk about some of the features of the Enterprise Edition (slide 3 of 3).

Workbook Section: Features of the Enterprise Edition

Visual Basic 4.0

Let's look at some of the important features in Visual Basic 4.0

The next few slides will be based on the list located on your workbook under the section title, *Visual Basic 4.0 Features List*. This is a quick slide. Make sure that everyone is caught up in the workbook, possibly open for some quick questions, and continue to the next slide.

Workbook Section: Visual Basic 4.0 Features List

Visual Basic for Applications (VBA)

- Visual Basic for Applications is the language in which you develop Visual Basic programs
- Compatible with Microsoft Access 95, Excel 5.0-7.0, and Project 4.0-7.0
- A full set of intrinsic functions and statements

Point out that the language syntax of Visual Basic 4.0 has been updated to support Visual Basic for Applications (VBA). Despite the language changes to Visual Basic 4.0, it is still 100% downward compatible with its forerunner, Visual Basic 3.0.

Visual Basic 4.0 can access the objects in Microsoft Access, Excel, and Project. All four of these products now share a common language, VBA.

The VBA Language contains many application specific functions that can assist the Visual Basic programmer in using all the tools available to them. For example, Microsoft Excel has special mathematical function for calculating rows of data while Microsoft Word has special text formatting functions.

Workbook Section: Visual Basic 4.0 Features List

Workbook Sub-section: Visual Basic for Applications (VBA)

VBA

- Conditional Compilation
 - Allows you to target 16/32-bit with one set of source code
- Built-in constants
 - Only those constants used are compiled into final .EXE file
- A complete set of data types
 - including Boolean, Date, and Byte

Conditional Compilation is not limited to separating 16-bit and 32-bit code segments - it could also be used to separate different languages (#IF L_FRENCH THEN) or debug mode (#IF DEBUG THEN)

With built-in constants, be sure to point out that this helps to protect you if Microsoft changes the values of these constants. It also benefits you if you are maintaining both a 16-bit and 32-bit applications because the compiler will use the appropriate constant based on your target operating system.

It is good practice to use constants whenever possible. It also assists in the readability of your code.

The three new data types were added to Visual Basic 4.0 to maintain compatibility with VBA.

Workbook Section: Visual Basic 4.0 Features List

Workbook Sub-section: Visual Basic for Applications (VBA)

ActiveX™ Controls

- Support for 16-bit VBXs and 16-bit ActiveX™ Controls in Windows 3.x
- 32-bit ActiveX™ Controls in Windows 95 and Windows NT
- 16-bit VBX support in Windows 95 & Windows NT via WOW
- Automatic morphing from VBX to ActiveX™ Controls

VBX's (Visual Basic eXtension) and OLE Custom Controls are two different types of containers for OLE controls. The VBX format was developed to use in older Visual Basic releases. They were typically written in C++, could only be accessed by Visual Basic, and were restricted to the 16-bit environment. OCX's can be written in a number of different languages and can be accessed by many different Windows applications, and can operate in both 16-bit and 32-bit environments. Both VBX's and OCX's are supported under Visual Basic 4.0.

Point out that 16-bit applications accessing 16-bit VBX's or 16-bit OCX's will still work under 32-bit systems like Windows 95 but will not perform as well as a comparable 32-bit application using 32-bit OCX's.

Be sure to point out that the automatic morphing of VBX's to OCX's does not mean that Microsoft provides a conversion utility that recompiles a VBX into an OCX. All this morphing means is that, each time you open a Visual Basic project that uses 16-bit VBX's, a search is made of the Registry to see if an identical OCX is registered on the system. If so, a dialog box is displayed asking the developer if they would like to change the reference of the VBX to point to the OCX instead.

Workbook Section: Visual Basic 4.0 Features List

Workbook Sub-section: OLE Custom Controls

ActiveX™ Support

- Over 140 third-party vendors are updating their VBXs to ActiveX™ Controls
- Support for Properties, Methods, and Events (VBX's do not support Events)
- Excellent OLE web site - <http://www.olebroker.com>

Point out that OCX's are here to stay. Microsoft has opened the OCX support to many Microsoft development languages and has given the third-party market a very large audience. It won't be long before all third-party tools are written in OCX's exclusively.

It is important to note that OCX's, unlike VBX's, now support Events. In addition to Properties and Methods, a developer can now add events to third-party OLE custom controls.

You can point out to students that <http://www.olebroker.com> is an excellent web page with tons of OLE controls and other reusable components. OLE Broker is published by ObjectSoft Corporation

Workbook Section: Visual Basic 4.0 Features List

Workbook Sub-section: OLE Custom Controls

ActiveX™ Support

- Reusable component that is available to programs such as Visual Basic 4.0, Microsoft Access 2.0/7.0, Visual FoxPro 3.0, and Visual C++ 4.x

Point out that, from a developer aspect, time spent learning a new OCX can now be applied to many developer languages. There is no need to learn one control that works with Visual Basic and another control that works with Microsoft Access. You can now carry that OCX (and knowledge) to many development languages.

Workbook Section: Visual Basic 4.0 Features List

Workbook Sub-section: OLE Custom Controls

Integrated Development Environment Features

- Tool-tips
- Right-mouse button everywhere
- Ability to lock controls
- Environment options allow you to customize your environment
- Can be controlled via OLE Automation by using Add-ins

This might be a good time to start Visual Basic and demonstrate some of these enhancements. There are no pre-defined demonstrations associated with this slide. I would recommend inventing your own demonstration or to simply ad-lib.

As a guideline, when demoing the tool tips, simply run you mouse over some of the buttons in the toolbar and show the students that tips really exist.

You can demo the right mouse click by using it on the toolbar or on the default form.

The remaining bullet points can be explained just as well without a demonstration.

Be sure to point out the benefits in the ability to customize the VBIDE. Explain that the Visual Basic environment is actually an OLE Server and that later in this presentation, a demonstration will show how we can link in our own menu options which allows us to add anything from version control to pasting pre-defined objects or controls.

Demo Scripts: Visual Basic IDE

Workbook Section: Visual Basic 4.0 Basics

32-bit ActiveX™ Controls

- Status Bar
- Slider
- Image List
- List View
- Tab Strip
- Tool Bar
- Rich Text Box
- Progress Bar

These controls are only available in the 32-bit edition of Visual Basic 4.0. These controls will assist you, as a developer, to maintain compatibility with the new Windows 95 standards of user interface design.

Demo Scripts: Using the Tab Dialog Control New Custom Controls

Workbook Section: Visual Basic 4.0 Custom Controls

Calling the Windows API

- Allows you to add functionality to your application above and beyond what Visual Basic 4.0 can provide.
- Is a risky decision because:
 - the Windows API is different between 16-bit and 32-bit operating systems.
 - Microsoft does not guarantee that the API functions will migrate into future releases of Windows.

Windows API stands for Windows Application Programming Interface. An API is simply a library of functions and the Windows API is simply a library of Windows functions. By calling these functions directly, you can perform tasks that Visual Basic may not provide an adequate means to accomplish. You can use the API to do such tasks as:

- Obtain information about Windows (version, memory free, resources)
- Obtain information about the system (type of CPU, Network capability)
- Execute object methods that Visual Basic didn't expose

These advantages do not come without risk. By using API calls, you may limit yourself to a particular version of Windows. API functions, unlike Visual Basic functions, are different in 16-bit operating systems and 32-bit operating systems. Also, Microsoft does not guarantee any particular API function will be carried over into the next release of Windows. Unlike Visual Basic functions, you may need to completely re-test your application when a new release of Windows comes out to insure that all your API functions act the same way they did in the previous version.

Visual Basic 4.0 does include conditional compile syntax (#IF...THEN) that will allow you to declare different API calls for different operating systems.

Workbook Section: Calling the Windows API

Class Modules

- Ability to create “stand-alone” Classes
 - Great for creating reusable components
 - Polymorphism
 - Encapsulation
 - Objects can be encapsulated within Objects
- Forms are now Callable objects
 - Custom Properties
 - Property Procedures

This is a major enhancement in the world of Visual Basic development. Visual Basic offers all of the benefits of object-oriented programming (OOP). You can point out that inheritance is the only characteristic of OOP that didn't make it into Visual Basic 4.0. This isn't a major setback because the primary benefits of OOP come from reusability, polymorphism and encapsulation.

“Reusability” is important because it gives the Visual Basic developer a way of writing modular code. Once a developer creates an object, that object can be re-used in other Visual Basic application, by other applications that act as OLE clients, and by other development languages.

“Encapsulation” and “polymorphism” let the programmer develop applications without needing to know the code behind the object. Encapsulation means that you can hide the variables and functions from the rest of the application. This is important to the concept of reusability. This is also beneficial to the developer who uses an object because they do not need to understand how the object works. This leads into polymorphism. Polymorphism allows two objects to share behaviors that are named the same but are implemented differently. For example, both the Form and a CommandButton have a Move method. As developer, you don't need to concern yourself with how the move, you just know that they will. If I introduce a class called MyControl and it also had a Move method, you would know that my code caused a behavior similar to the Form and CommandButton. That's polymorphism.

Regarding forms, point out that, unlike previous releases of Visual Basic, forms can now be treated like objects. You can access the variables of a form just like you access properties of an object. You can define custom methods for forms that other code modules or objects can call.

A simple demonstration of class implementation will be shown later in this presentation.

Do Demonstration: Using Classes

Workbook Section: Class Modules

Internet Control Pack

- Point 1
- Point 2
- Point 3
- Point 4

{Slide left intentionally blank}

OLE Servers

- Ability to create OLE Servers in Visual Basic
 - Control these servers from any OLE application
- Remote OLE Servers
 - Run the OLE server on any machine on a network

The ability to create OLE servers is an important feature in Visual Basic 4.0. OLE Servers created with Visual Basic 4.0 can be accessed by any OLE Client Application including Microsoft Access 95, Microsoft Word, Microsoft Excel, and Microsoft Visual C++.

Remote OLE Servers are also very powerful, especially in a three-tier client/server architecture which we will be talking about later in this presentation. What a remote OLE server allows me to do is to have a Visual Basic application communicating across a wide area network. For example, I could allocate a network machine to perform all business rules for my network application. All clients would access my business rules via OLE across the network. If I need to change the business rules of my application, I need only change the OLE server application on ONE machine.

This topic will be demonstrated and talked about more later in this presentation.

Add-ins

- Add-ins menu allows third-party vendors to add functionality to the IDE
- Can write Add-ins in Visual Basic or Visual C++
 - Full object hierarchy exposed in IDE
 - Can control via OLE Automation

The Visual Basic programming environment can be controlled via OLE Automation. This means that Visual Basic environment is actually a regular OLE object containing properties, methods, and events. Within the Visual Basic documentation, there is a document list of all the object exposed to you, the developer. Visual Basic allows you to access methods that permit you to add menu options as well as have that menu option call a method within your Add-In Object. Visual Basic has opened a large door to developers to completely customize their programming environment. Now you can add menu options that cut and paste code, provide version control, provide a user-friendly database interface, and much more.....

Do Demonstration: Add-Ins

Workbook Section: What Is An Add-Ins?

Other Enhancements

- Integrated, intelligent setup wizard
 - Network setups and uninstall
- Special compiling features
- Resource files can be used to create multi-lingual versions of your applications
- SourceSafe Version Control

The new setup wizard now supports Windows 95 long filenames, uninstall applications, 16-bit and 32-bit versions, and support for client/server installation by registering OLE servers with the Windows Registry.

Visual Basic 4.0 supports "Compile on Demand" which will only compile portions of the code if it needs to be executed. Visual Basic 4.0 also provides support for "Background Compile" which, during run-time, will compile portions of the code during idle CPU cycles.

Visual Basic now supports resource files that allow you to internationalize your application. With resource files, you can store text and images based on the desired language or desired audience.

The Enterprise Edition of Visual Basic 4.0 includes a full copy of Microsoft SourceSafe which a developer uses as version control software. SourceSafe integrates with Visual Basic as an Add-In and allows the "checking-in" and "checking-out" of Visual Basic modules from a centralized server.

Do Demonstration: SourceSafe

Workbook Section: Version Control

Integrated Jet 3.0 Database Engine

- Jet 2.5 (16-bit) and Jet 3.0 (32-bit) engines with full database object programmability
- Full database creation from Visual Basic code
 - Including referential integrity
- Programmable Replication support with Jet 3.0
- Data-aware controls
 - including List Box, Combo Box, and Grid

Both Visual Basic 4.0 and Access 95 now share a common Jet Engine. Visual Basic ships with two Jet Engines: a 16-bit (Jet 2.5) and a 32-bit (Jet 3.0). Point out that 16-bit Jet Engine can only be used with the 16-bit edition of Visual Basic, likewise the 32-bit Engine can only be used with the 32-bit edition of Visual Basic. The 32-bit Edition does provide an additional database object that allows developers of 32-bit application to access databases created using Jet 2.5. Obviously, the reverse of that statement cannot be true.

Visual Basic developers can now create Microsoft Access 7.0 databases with full referential integrity. Essentially, anything that can be created, defined, or deleted using Microsoft Access 7.0 can be done using Visual Basic 4.0.

Database Replication allows developers to write a multi-user application that can access copies of the main database. The Jet Engine will propagate all changes done to the main database to the copies and visa-versa. By allowing you to keep multiple copies of the data, you can decrease the load on a local network, keep better backups, and provide data synchronization over a wide area network.

Visual Basic has numerous data-aware controls that allow developers to create powerful database application with little or no coding.

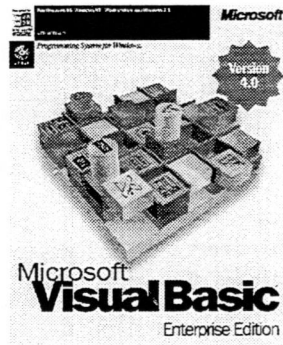
The information on this slide will be demonstrated and serves as a transition slide into the next section, "Database Development".

Visual Basic 4.0 Features

Questions and Answers

Take time now to answer student questions.

Database Development



This next section will take a look at five different methods of accessing data using Microsoft Visual Basic 4.0. Point out that you will discuss each of the methods and their benefits and disadvantages. You will also demo each of the methods and you can tell the audience that the source for each demonstration is on their sample disk.

This is section 2 of 4 and should last approximately two hours.

Workbook Section: Introduction

Data Access Methods

Method	Edition Available

This slide gives you an overview of the five different methods that are available using Microsoft Visual Basic 4.0.

The Data control is available in all editions. The Data Access Object is available only in the Microsoft Visual Basic Professional and Enterprise Editions. If you are a developer going against a Microsoft Access database, these methods are the best methods to use.

The Remote Data control and the Remote Data Object are only available in the Microsoft Visual Basic 4.0 Enterprise Edition and can be compiled only in 32-bit. These methods are the best methods to use against a client/server back end such as Microsoft SQL Server, Oracle, Sybase, etc.

The ODBC API is available in all editions. This is a very fast method of database access, but it requires a lot of knowledge about the ODBC API functions, it requires a lot more coding, and it is risky because Microsoft does not guarantee that the ODBC API will not change in future releases.

Workbook Section: Database Programmability

Data Control (DC)

- Control-level interface
- Very easy to use
- Many built-in data-aware controls
- Little coding required to create an Add, Edit, and Delete form

We start off discussing the Data control. Point out that this is a control-level interface to the Microsoft Jet 2.5/3.0 Engine. Although the programmer is limited in flexibility by the number of properties and methods, this method is very easy to learn and very easy to use. In Visual Basic 4.0, Microsoft has included many data-aware controls including TextBoxes, ListBoxes, ComboBoxes, and even a data-aware Grid control.

DC - Important Properties

- Connect
- Database
- DatabaseName
- Recordset
- RecordsetType
- RecordSource
- BOFAction, EOFAction

Quickly run through some of the important properties giving a brief explanation of each.

Connect property contains information about the source of the database. Primarily used to set specific information required by ODBC drivers. If you are connecting to a Microsoft Access database, this property is ignored.

Database property is only available in the Professional and Enterprise Editions and is used to get a reference to the underlying Database object that the Data control is using.

DatabaseName property is used to set the name and path of the database file.

Recordset property is used to get or set the Data control's current Recordset Object.

RecordsetType property is used to define the type of RecordSet the Data control will use (i.e. Table, Dynaset, or Snapshot).

RecordSource property is used to set the underlying table, SQL statement, or QueryDef object for the Data control.

Demo Script: Data Control

Workbook Section: Using The Data Control

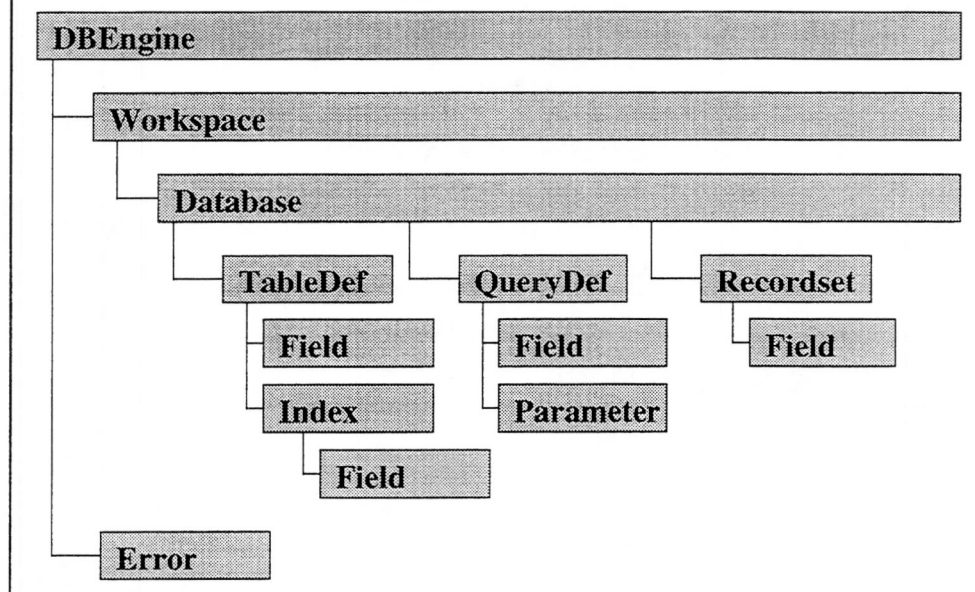
Data Access Objects (DAO)

- Language-level interface
- Available in Microsoft Visual Basic Professional and Enterprise Editions only
- Coding required to create an Add, Edit, and Delete form
- Full control over recordset objects

Point out that Data Access Objects (DAO) provide a language-level interface. They are not dependant on any control and all manipulation of the object has to be done via coding. Unlike the Data control, DAO is only available in the Microsoft Visual Basic 4.0 Professional and Enterprise Editions.

As the audience will see in a minute, to do normal adding, editing, and deleting, more coding will be required. The advantage of doing all the coding yourself is that you gain more control and flexibility.

DAO Object Model



This is a simple chart of the DAO Object. Please point out to the students that this chart is NOT all inclusive—some objects have been omitted.

The DBEngine is the highest object in this hierarchy and represents the Microsoft Jet Engine.

Each DBEngine has a collection of Workspace objects. The benefit of a Workspace is that it provides for simultaneous transactions and secure workgroups.

Each Workspace has a collection of Database objects that represent each physical open database.

Each Database can have collections of TableDef objects, QueryDef objects, and Recordset objects.

Point out to the students that this hierarchy provides the programmer with a very powerful database object, and a major benefit of using objects is that they are not back end specific. The same object coding will work against a Microsoft Access table, a SQL table, or even a FoxPro table.

Demo Script: Data Access Objects

Workbook Section: Using the Data Access Objects

ODBC API

- Requires extensive knowledge of ODBC
- Must understand call-level interfaces
- Coding required to create an Add, Edit, and Delete form

Explain to the audience that using the ODBC API is an advanced way of accessing data. There are limitations and risks involved with using it. The point of reviewing it is to educate the students as to what it is and how it works so that they can make an informed decision as to whether it is right for them.

Using the ODBC API requires an extensive amount of up-front knowledge about how ODBC works.

You must have a good understanding of call-level interfaces. This means that the programmer must be comfortable with making direct calls to .DLL files. Once you start making direct calls, you leave the protective shell that Visual Basic provides. All error handling must be done by the programmer and you run a higher risk of incompatible programs or, worse yet, crashing Windows.

Lastly, programming with the API requires more coding than any other method you will see here today.

The advantage of using the ODBC API is speed. You have cut out many layers between you and your data.

Workbook Section: ODBC

Purpose of ODBC

- A set of API calls that allow you to connect to multiple back ends
- A translator of SQL statements

The question here is, “Why use ODBC?”

ODBC stands for Open Database Connectivity. ODBC libraries provide data-access connectivity to Microsoft SQL Server, and any other database that provides an ODBC driver. By coding to this interface, you can create database-independent code, which means the ODBC API provides a universal programming model that automatically adapts to a variety of databases. Therefore, in theory, an application designed to communicate with an ODBC driver, say for Microsoft SQL Server, can be converted to another SQL back end by simply changing the ODBC driver—without making any coding changes.

To look at the ODBC driver in simple terms, the driver converts ODBC standard SQL statements into SQL statements that are supported by the back end.

Workbook Section: Purpose of ODBC

Microsoft Access SQL

Access SQL Syntax:

```
SELECT tblCustomers.sCompany_nm,  
       tblInvoice.szInvoice_desc  
FROM   tblCustomers LEFT OUTER JOIN  
       tblInvoice  
ON     tblCustomers.lCust_id =  
       tblInvoice.lCust_ID
```

These next three slides show three different SQL standards. These slides are intended to give the audience an idea of what each is like, and to show that trying to learn all SQL standards is something that can be avoided by going to a ODBC SQL standard.

The first SQL standard is Microsoft Access SQL, which is the only SQL supported by the Microsoft Jet Engine. If you use the Data control or the Data Access Object, all your SQL statements must conform to the Microsoft Access SQL standards.

Microsoft SQL Server

SQL Server Syntax:

```
SELECT tblCustomers.sCompany_nm,  
       tblInvoice.szInvoice_desc  
FROM   tblCustomers, tblInvoice  
WHERE  tblCustomers.lCust_id *=  
       tblInvoice.lCust_id
```

The next slide shows a typical back end SQL standard and, in this case, we are showing you the Microsoft SQL Server standard.

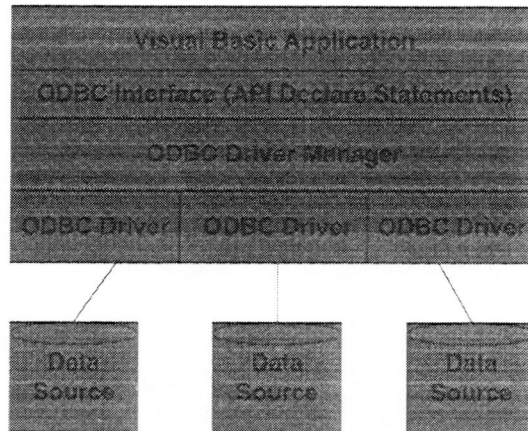
ODBC SQL

ODBC Syntax:

```
SELECT tblCustomers.sCompany_nm,  
       tblInvoice.szInvoice_desc  
FROM { oj tblCustomers LEFT OUTER JOIN  
       tblInvoice  
ON tblCustomers.lCust_id =  
   tblInvoice.lCust_id }
```

The last of these three slides is the ODBC SQL. If you learn the ODBC SQL language, you can access any SQL back end that has an ODBC driver.

ODBC Components



This is a diagram of the components that are necessary in order to use the ODBC API. The explanation of each level is as follows:

First we have the Visual Basic application.

ODBC API Interface (API Declare statements) expose specific ODBC API functions to Visual Basic and the procedures that need to access the API. Visual Basic applications use the APIs like any other function.

The ODBC driver manager provides the interface from the host language to the specific back-end data source driver. The driver manager is responsible for:

- Loading the remote database driver specified in the data source name entry.
- Initializing the interface.
- Providing entry points to driver entry points.
- Validating parameters and managing serialization of ODBC functions.

The ODBC driver is a dynamic link library (DLL) that can interface with a specific back-end database engine or a specific type of file implemented as a database. For example, there are specific drivers that connect to Microsoft SQL Server and other drivers that connect to Sybase SQL Servers, Oracle, or Microsoft Visual FoxPro databases.

Workbook Section: ODBC Components

ODBC Setup

- Use the ODBC Administrator
- Create a DSN Name
- Specify a database

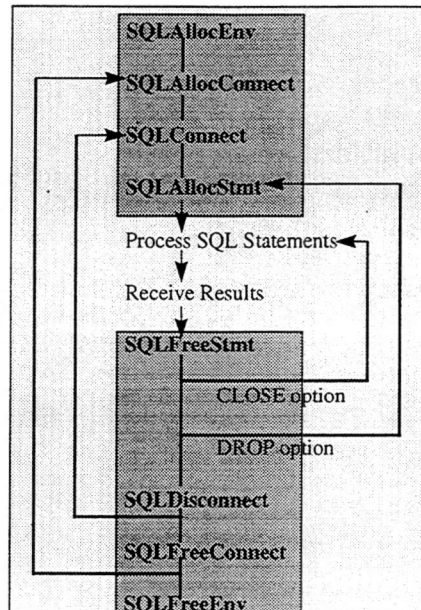
In order to use the ODBC driver in Windows, you need to do some initial setup using the Microsoft ODBC Administrator program. This application comes with Microsoft Visual Basic 4.0 Professional and Enterprise Editions as well as many other Microsoft products.

After you execute the ODBC Administrator, all you need to set up a link to your database is a DSN (Data Source Name) and a hard-coded path to either the data itself or the SQL database engine.

Demo Script: ODBC Setup

Workbook Section: ODBC Setup

ODBC Function Overview



The above figure shows the core ODBC API functions you use to connect to a remote server, submit an SQL query, retrieve the results, and disconnect from the server.

Virtually all ODBC API applications use the core set of API calls to perform fundamental operations. All ODBC applications must establish an environment handle (**SQLAllocEnv**), a connection handle (**SQLAllocConnect**), and a statement handle (**SQLAllocStmt**). These handles provide addressability to structures created by the ODBC driver manager and drivers that are used to store parameters, errors, and returned arguments.

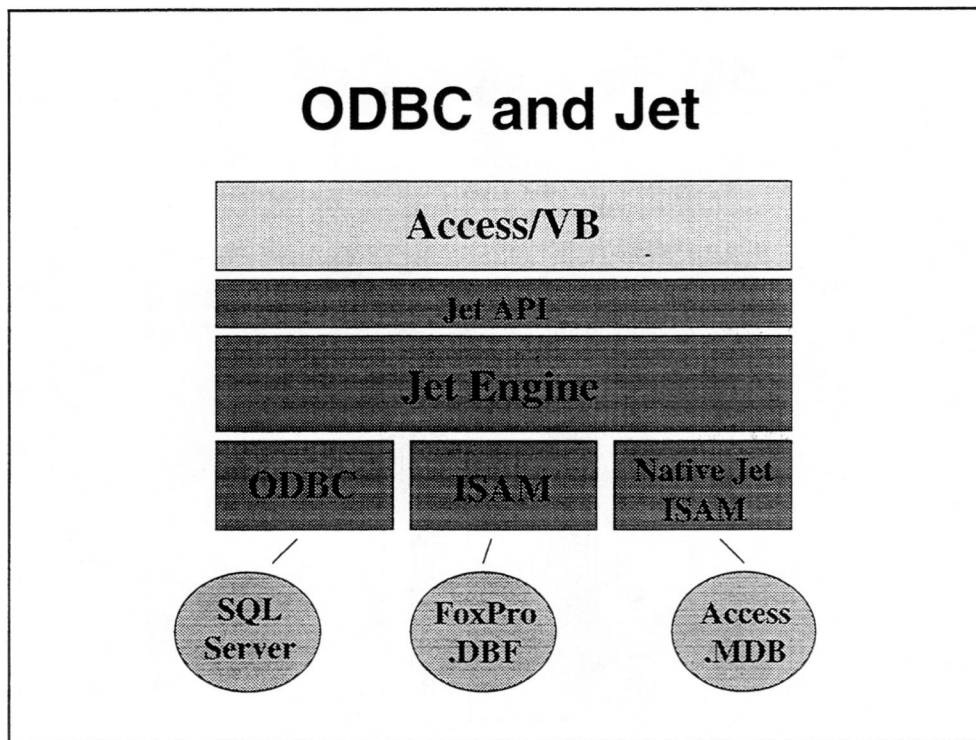
Once a connection is established (**SQLConnect**), SQL statements are then executed (**SQLExecDirect**) by the ODBC driver. Results from the query are processed (**SQLFetch**), data values for each column are retrieved (**SQLGetData**), and the process is repeated as needed. Freeing the statement handle (**SQLFreeStmt**) releases the result set retrieved by the query. Once the connection is no longer needed, it is closed (**SQLDisconnect**). When access to data is no longer needed, the connection and environment handles are released (**SQLFreeConnect**, **SQLFreeEnv**).

This section may go over the heads of many students, but it gets the point across about how much is involved in doing simple data manipulation.

Demo Script: ODBC API

Workbook Sectin: Using the ODBC API

ODBC and Jet



The diagram above shows how Jet works in conjunction with ODBC. Discuss that using Jet to access ODBC is much easier to program and eliminates many of the downsides that are present in directly calling the ODBC API.

This slide serves as a lead in to the Remote Data control, which is a Visual Basic object that is a true wrapper for the ODBC API. The Remote Data control is only available in the Microsoft Visual Basic 4.0 Enterprise Edition, so people who have just the Professional Edition may still want to investigate using Jet with ODBC.

Remote Data Control

- Control-level interface
- Very easy to use
- Many built-in data-aware controls
- Little coding required to create an Add, Edit, and Delete form

The Remote Data control (RDC) is an example of yet other data access method. The RDC binds the Remote Data Object (RDO) to a control in much the same way that the Data control binds the Data Access Object (DAO) to a control. The RDO is a thin layer of code placed on top of Open Database Connectivity (ODBC) and handles making all the necessary ODBC API calls.

Explain to the audience that RDC are most advantageous when running against ODBC to a SQL back end. RDC is not a good substitute for a Data control when going against a Microsoft Access database (in fact, it will most likely be slower).

Make sure that the audience understands that this is a control-level interface and that the programmer's control over the data is limited by the properties and methods of the Remote Data control.

To better prepare yourself for the Remote Data Control and Remote Data Object, please refer to the article by Ken Lassesen called "Introduction to Using the Remote Data Object" (Sept 11, 1995).

[This is on the TechNet CD, Darrique. Can we include it on the CD?]

Workbook Section: Remote Data Control

RDC - Important Properties

- Connection
- Connect
- CursorDriver
- DataSourceName
- MaxRows
- Password
- ReadOnly

Before going into the Remote Data control demo script, it would be good to give a brief explanation of the following properties:

Connection: A reference to the underlying rdoConnection object.

Connect: A valid connect string as determined by the ODBC Driver. For example, you may pass the DSN, UID, PWD, or DATABASE to the Microsoft SQL Server Driver.

CursorDriver: May set to one of three values. You may specify for the ODBC driver to determine which cursor mode to use. You may specify to use the ODBC Cursor Library. Or you may specify to use Server-side cursors available with Microsoft SQL Server 6.0 or Oracle 7.0.

DataSourceName: May be left blank if the DSN is filled in using the Connect property (property used primarily for Microsoft Access databases).

MaxRows: May be set to the maximum amount of rows to return from a query.

Password: May be used if the PWD option in the Connect property is left blank.

ReadOnly: Returns TRUE if the underlying cursor is read-only.

Workbook Section: Properties of the Remote Data Control

RDC - Important Properties

- Resultset
- ResultsetType
- RowsetSize
- SQL
- Transactions
- UserName

Resultset: Returns a reference to the underlying rdoResultset object. May also be used to set an existing rdoResultset to the Data control.

ResultsetType: Returns or sets the type of cursor that is/will be opened. You may specify a 1 for a Keyset type. This is an editable cursor, but any changes made in the cursor show up as holes in another user's cursor. You may specify 3 for a cursor that is similar to a snapshot object, but is updateable. Changes made to this type of cursor are not reflected in other user's cursors.

RowsetSize: The number of rows of data that will be buffered on the client machine after a cursor is opened. The default is 100.

SQL: The SQL statement that will be passed to the data source for processing. This may be a back-end specific SQL statement or one with ODBC escape clauses in it.

Transactions: Returns True if the data source supports transaction processing.

UserName: Set this property if you have not used the UID in the Connect property.

Demo Script: Remote Data Control

Workbook Section: Properties of the Remote Data Control

Remote Data Objects

- Language-level interface
- Requires knowledge of ODBC
- Full control over resultset objects
- Coding required to create an Add, Edit, and Delete form

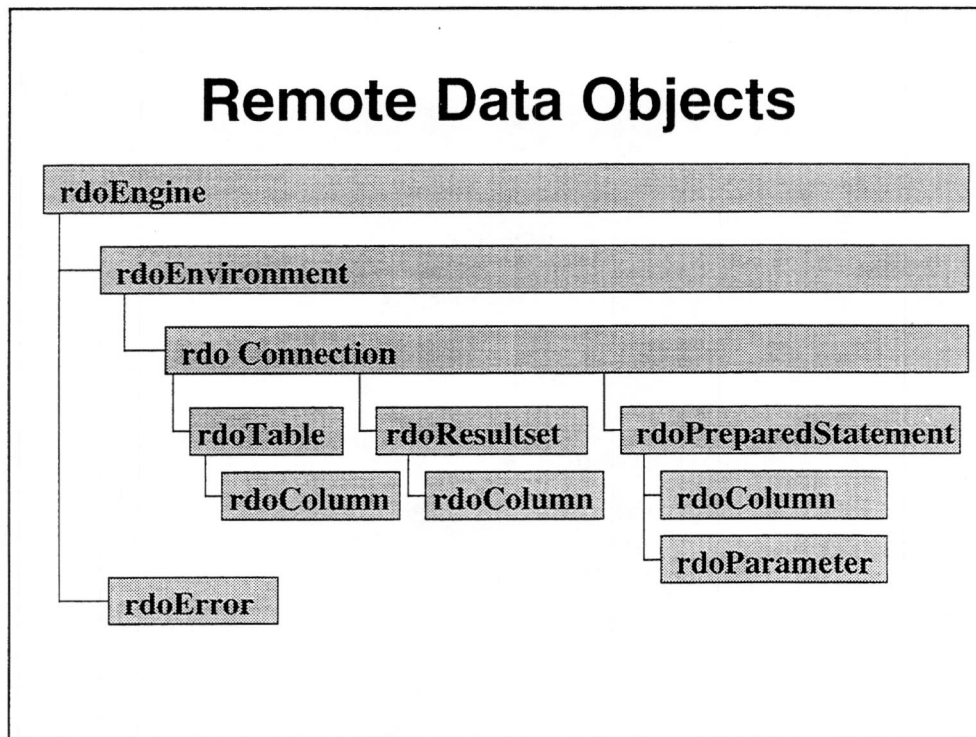
Point out that this is similar to DAO except that it is going against ODBC instead of Jet. This is a language-level interface to all data access and must be done through coding. Working with the Remote Data Object requires more knowledge of ODBC than the Remote Data control.

With RDO, you have more control of the resultset object since you do not have to go through a control to get at the object's properties and methods.

And, like DAO, you will need to do more up-front coding in order to perform the basic Add, Edit, and Delete functions.

Workbook Section: Remote Data Access Objects

Remote Data Objects



This is a diagram of the Remote Data Access Object hierarchy. Use this diagram to point out the similarities of it to the Data Access Object. For example, instead of DBEngine, we have rdoEngine; instead of Workspace, we have rdoEnvironment. This should help students understand the next few slides that go into the properties of the rdoEnvironment, rdoConnection, and the rdoResultset.

Wordbook Section: RemoteData Access Objects

RDO - Important Properties

- **rdoEnvironment**
 - **hEnv**
- **rdoConnection**
 - **hDbc**
 - **RowsAffected**
 - **CreatePreparedStatement**
 - **Execute**
 - **OpenResultSet**

rdoEnvironment object: This object contains a logical set of connections for a particular user ID. It contains open databases, provides mechanisms for simultaneous transactions, and provides a security context for data manipulation language (DML) operations on the database.

- *hEnv*: The underlying ODBC Environment handle.

rdoConnection object: This object represents an open connection to a remote data source and a specific database on that data source.

- *hDbc*: The underlying ODBC database connection handle.

- *RowsAffected*: The number of rows affected by the last INSERT, UPDATE, or DELETE statement.

- *CreatePreparedStatement*: Allows you to create a prepared SQL statement that can be called over and over again with replacement parameters. Think of this as a temporary stored procedure that accepts parameters.

- *Execute*: Executes an action query such as an INSERT, UPDATE, or DELETE statement.

- *OpenResultset*: Creates an rdoResultset object.

Workbook Section: Properties of the Remote Data Objects

RDC - Important Properties

- rdoResultSet

- GetRows
- MoreResults
- hStmt
- Restartable

rdoResultSet object: This object is the set of rows that results from running a query.

- *GetRows*: Returns a set of rows into a two dimensional array.
- *MoreResults*: Clears the current result set and returns a true if there are additional result sets to be processed.
- *hStmt*: The underlying ODBC statement handle.
- *Restartable*: Returns a true if the resultset supports the Requery method.

Demo Script: Remote Data Objects

Workbook Section: Properties of the Remote Data Objects

Advantages/Disadvantages

- Using Jet Data Control
 - Very easy to learn
 - Lower performance on ODBC data sources
- Using Jet Data Access Objects
 - Slightly higher learning curve
 - Better performance than Data Control

Treat these next three slides as a quick compare and contrast as well as a little bit of a review. After touching on each method's pros and cons, you can go straight into the Q & A portion.

Advantages/Disadvantages

- Remote Data Control
 - Easy to learn
 - Requires minimal knowledge of ODBC
 - Good performance on ODBC data sources
- Remote Data Objects
 - Higher learning curve
 - Requires minimal knowledge of ODBC
 - Good performance on ODBC data sources

Advantages/Disadvantages

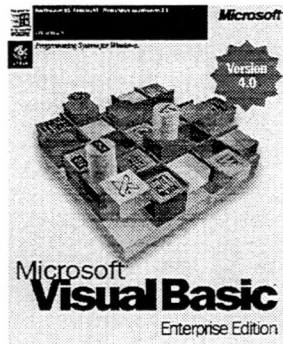
- **ODBC API**

- Can utilize all functionality of ODBC
- Great performance
- Higher learning curve
- Microsoft does not guarantee that functions will operate the same in future releases of the ODBC API.

Database Development

Questions and Answers

Application Framework



Developing a good application framework is not a requirement with Visual Basic. Furthermore, there are as many different ways to design a framework as there are developers making applications. Therefore, the adaption of an application framework should not be done quickly, but instead should be carefully thought out and agreed upon by all members of a development team.

What these next few slides show is one interpretation of how an application framework could be assembled. This framework design is not the only design nor is it necessarily the best design for all situations but it should help in giving developers a good start at developing their own application framework.

Why have a Framework?

- Provides a starting point for all applications
- Standard design for all business objects
- Back end independant
- Better prepared for change
- Decreases development time

An application provides a good starting point for all applications. If you develop a standard template application that you can use for all your development projects, you are essentially creating reusable components. It is easy to see how this can save time and cut down on testing and debugging your application.

By having development team developing components based on a standard design, it is easier to solve business solutions by simply assembling many interchangeable components.

As you create a data-independent framework, you can better prepare yourself for changes. If an application written to run against a Microsoft Access database suddenly needs to be upgraded to run against Microsoft SQL Server, the better you were at creating data independence, the easier it is to migrate your application to a different back end.

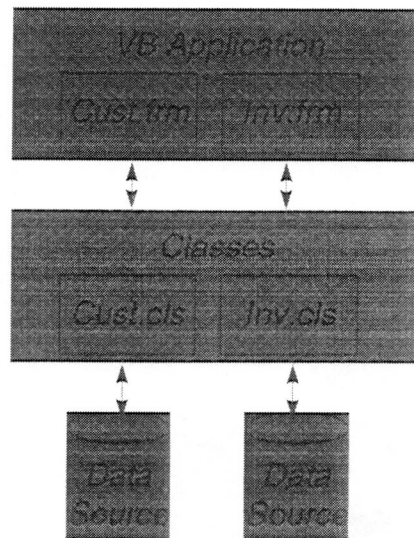
With a good framework, the impact to an application because of changes to the specifications can be minimized. In theory, objects within a project can be removed, swapped, rewritten, etc. without affecting other objects in the projects. This allows developers to make changes without affecting other areas of the application.

Objectives For An Application Framework

- Encapsulate data access routines in class modules
- Create standard methods for all objects
- Use objects.properties only in Forms
- Do not use any Visual Basic controls in data access classes

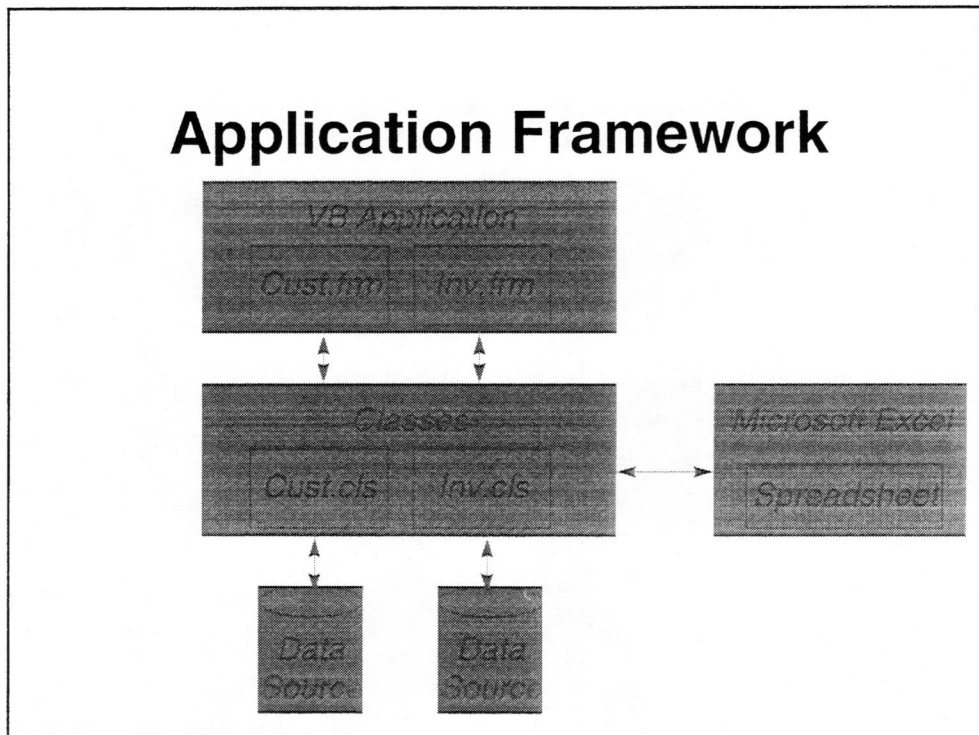
One goal of using an application framework is to separate the user interface code from the data access code. The best way to accomplish this is to maintain classes for data access, classes for business rules, and classes for general user interface. Once you have your classes designed and created, you can program for forms using the {object}.{property} method and allow yourself the freedom to swap different back-end classes without changing the forms code (this is an excellent example of polymorphism). If the "Add" button on a form makes the call `MyDatabase.AddRecord`, the object `MyDatabase` can be Microsoft Access or Microsoft SQL Server because the method will respond differently depending on the class.

Application Framework



Here's a graphical look at what was described in the previous slide. Here you see our forms CUST.FRM and INV.FRM and our classes CUST.CLS and INC.CLS which access a particular data source. It is easy to see how by swapping out the classes with another set, you could theoretically change back ends without changing any of the code in the Visual Basic application.

Application Framework



Another benefit of moving the data access into their own classes is that any OLE client can use them. Later in this presentation, we will demonstrate how we can use Microsoft Excel to send data to a Microsoft Access table using our data access classes.

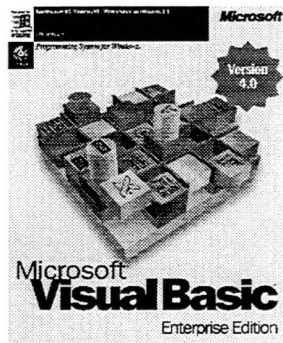
Now, let's take a look at a demonstration of using this application framework.

Demo Script: Application Framework

Application Framework

Questions and Answers

OLE Servers



This is the fourth of four sections within the Visual Basic 4.0 presentation. This section covers OLE, OLE Servers, and OLE Automation. This section should take about 1.5-2.0 hours.

OLE Servers

- What is an OLE Server?
 - An application that can be controlled by another application
- What is an OLE Client?
- How do you create an OLE Server?
 - Create Class Modules
 - Expose Properties and Methods

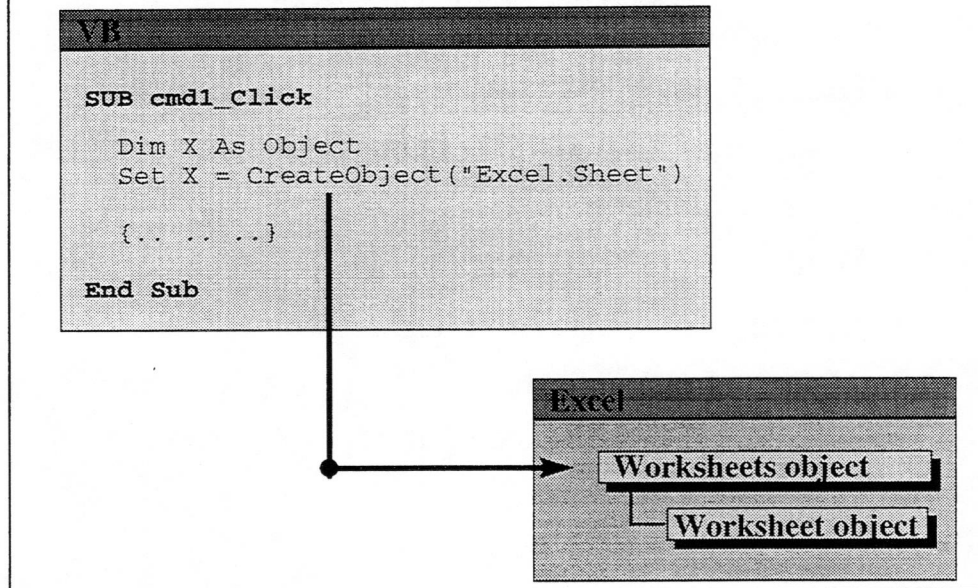
Point out that an OLE server is simply an application that can be controlled by another application. That controlling application is normally called the OLE Client. As an example, Visual Basic 4.0 is an OLE Server. So is Microsoft Access 95, Excel 95, Word 95, and many others.

When developing OLE Servers with Visual Basic 4.0, make sure to point out that they can only run in a 32-bit operating system. The only way around this rule is through Remote OLE which allows 16-bit applications to access OLE Servers on 32-bit operating systems over a network connection.

Point out that creating OLE Servers is as simple as creating classes using Visual Basic 4.0. Classes that you create in Visual Basic 4.0 can be compiled into OLE Servers and then accessed by other Visual Basic 4.0 application, Microsoft Excel 95, Microsoft Visual FoxPro, or any OLE Client.

Workbook Section: OLE Servers

OLE Servers Example



The above is a quick diagram of a Visual Basic 4.0 application creating an Excel object. Point out to the audience that the two lines above will cause Excel to load in the background and the menu options “File...New” to be invoked. This is no different than if you manually went into Excel via the Program Manager and pressed “File...New” yourself.

Point out that we will see an example of this shortly.

OLE Server Example

```
Private Sub Cmd1_Click()  
Dim X As Object  
  
Set X = CreateObject("Excel.Sheet")  
  
X.Cells(1, 1).Value = Text1.Text  
X.Cells(2, 1).Value = Text2.Text  
  
X.Cells(3, 1).Formula = "=R1C1 + R2C1"  
Text3.Text = X.Cells(3, 1)  
  
X.Application.Quit  
Set X = Nothing  
End Sub
```

- Line 1 declares "X" as an object.
- Line 2 assigns "X" to an Excel object.
- Line 3 assigns the value in the TextBox Text1 to the Cell (1,1).
- Line 4 assigns the value in the TextBox Text2 to the Cell (2,1).
- Line 5 creates a formula in Cell (3,1) which adds Cell (1,1) to Cell (2,1).
- Line 6 assigns the value in Cell (3,1) to TextBox Text3.
- Line 7 causes Excel to unload from memory.
- Line 8 assigned the variable "X" to nothing which removes it from memory.

Demo Script: Excel Calculator

Three-Tier Architecture

User Services



- Client

Business Services



- Client
- Server
- Data Server

Data Services



- Data Server
-

One very important use for OLE Servers is in three-tier client/server architecture. This architecture separates the various components of a client/server system into three “tiers” of services that must come together to create an application.

User services: Presentation of information and functionality, navigation, protection of user interface consistency and integrity.

Business services: Shared business policies, generation of business information from data, protection of business integrity.

Data services: Definition of data, storage and retrieval of persistent data, protection of data integrity.

In a network situation, the client machine (or user machine) is typically only running 1st tier services; likewise, your data server (server running your back-end software) is typically only running 3rd tier services.

The 2nd tier services (OLE server) can be one of three places. It can be on the client machine. The main drawback is that updated business rules must then be propagated over all clients. The OLE server can be on the data server. This limits you to only one machine that you need to modify if your business rules change, but the OLE server has to share CPU processing time with the back-end software. The best solution is to dedicate its own server so that you isolate the business rules and your OLE server does not have to share CPU processing time.

The next three slides will look at each tier individually.

Workbook Section: Three-tier Client/Server Architecture

Three-Tier Architecture

- Tier 1

- User services
- Display/Retrieve information
- Calls the 2nd tier passing information
- Created with a front end tool such as Visual Basic 4.0, Microsoft Access, Visual FoxPro, etc.



The user services tier provides the user interface for presenting information and gathering data. Basically, they integrate the user with the application to perform a business process. User services are generally identified with the user interface, and normally reside in an executable program located on the end user's workstation.

Point out that user services, by definition, do not allow any data access. Refer back to the demo used during the Application Framework section and point out that the Form modules and general code modules would be a good example of what the user services would be. User services make calls to class modules which represent the second tier, or business services.

Workbook Section: Three-tier Client/Server Architecture

Three-Tier Architecture

- Tier 2

- Business services
- Validates data
- Submits data to the back end database
- Receive data and error messages from the back end database
- Communicates between Tier 1 & 3



The second tier represents business services—the “bridge” between user and data services. They respond to requests from the user (or other business services) in order to execute a business task. This insulates the user from direct interaction with the database.

A business task is an operation defined by the application’s requirements, such as entering a purchase order or printing a customer list. Business rules are policies that control the flow of the business tasks. Because business rules tend to change more frequently than the specific business tasks they support, they are ideal candidates for encapsulating in components that are physically separate from the application logic itself.

Point out that this tier is similar to the Jet and ODBC classes that you discussed in the Application Framework demo. The second tier classes can be compiled into an OLE server, our customer application (OLE client) can be compiled into an executable (EXE), and they can communicate with one another via OLE. They can even communicate over the network. To take this point one step further, you could have one copy of the OLE server running on a Windows NT machine and have twenty Windows 3.11 users running your customer EXE communicating with this one OLE server. Imagine how easy it would be to update all 20 users to communicate with Microsoft SQL server instead of Microsoft Access.

Workbook Section: Three-tier Client/Server Architecture

Three-Tier Architecture

- Tier 3
 - Data services
 - Responsible for managing/storing data
 - Inserting, updating, and deleting rows
 - Maintaining referential integrity
 - Third tiers are any back-end databases such as Microsoft SQL Server, Oracle, Sybase.



Our third tier represents our data services. These services define, maintain, access, and update data, and manage and satisfy business services' requests for data. They may be physically implemented in a particular database management system (DBMS), or by a heterogeneous collection of databases that may reside on multiple platforms and combinations of servers and mainframe computers.

In our customer application example, this tier would represent the Microsoft Jet engine or Microsoft SQL Server.

Workbook Section: Three-tier Client/Server Architecture

Business Rules

- What do you put in the 2nd tier?
 - Business rules
- Which rules do you put in the 1st tier?
 - Rules that provide immediate feedback to user
 - However if rules change, all applications that use those rules must change and be redistributed

These next few slides will take a look at business rules. First point out that business rules reside in the 2nd tier (business services).

The only business rules that should be placed in the 1st tier (user services) are those that need to provide immediate feedback to users. When determining your 1st tier rules remember two things:

1. Any rule that is in your 1st tier should also be duplicated in your 2nd tier. This will prevent users from bypassing your application and going straight to your OLE server and entering bad data. (Remember, users can get at your objects via any OLE client like Microsoft Excel.)
2. Any changes in these rules will require you to go back and redistribute a new EXE to all clients.

It is easy to see why only a few rules should be allowed in your 1st tier.

Workbook Section: Business Rules

Business Rules

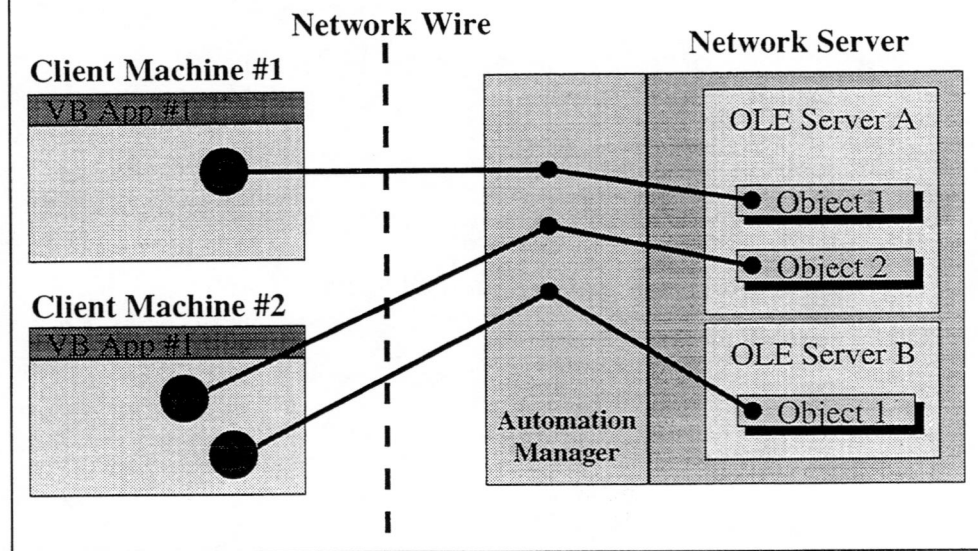
- Placing rules on the back end (3rd tier)?
 - Centralized
 - Enhanced security
 - Limited to SQL language
 - Not great for complicated business rules

All 2nd tier rules need to be duplicated on the 3rd tier if at all possible. It is nearly impossible to prevent your users from bypassing both your application and your OLE server and going straight to your data source. Any rules that you add to the 3rd tier will greatly enhance your applications integrity.

Point out that 3rd tier rules are generally limited to the SQL language so being able to translating 100% of all business rules is very rare.

Workbook Section: Business Rules

Remote OLE Servers



This diagram above shows two Visual Basic 4.0 applications running on two separate computers on a network. Client machine #1 has an application that makes a request for an instance of OLE Server A. Through the use of the Remote Automation Connection Manager on the Client machine, the OLE server reference is set to point to a network server. The Visual Basic application then contacts the Automation Manager, located on the network server, requesting an instance of OLE Server A. The Automation Manager is responsible for creating the instance and managing the interaction between the Visual Basic application and the OLE server.

Also in this picture is Client Machine #2. This machine has an application that requests an instance of the OLE Server A and an instance of OLE Server B. As you can see, the Automation Manager manages these requests as well.

Workbook Section: Remote OLE Servers

OLE Servers

Questions and Answers

This ends the PowerPoint presentation. There are now four OLE server examples available to you that you can choose one or more of to conclude this presentation with.

Demo Script: OLE Server Example #1

Demo Script: OLE Server Example #2

Demo Script: OLE Server with Microsoft Excel

Demo Script: OLE Server Example #3

Remote OLE Servers

Another exciting technology available with Visual Basic 4.0 is the ability to take the OLE servers you create, put them on any machine on the network, and have users from any client machine run those OLE servers over the network! The OLE server actually runs on the machine on which it resides and just sends the object information back to the client machine. This makes it easy to implement a central repository for your business rules. It also lets you leverage the horsepower of one powerful machine for running the OLE server with the client machines that do not need as much horsepower.

```
Sub SaveCustomer()  
    Dim oCust As Object  
  
    On Error GoTo SaveCust  
  
    Set oCust = CreateObject("CustData.clsCust")  
  
    ' Set adding flag  
    oCust.bAdding = True  
  
    ' Load customer data  
    oCust.lCustID = 0  
    oCust.sCompany =  
Worksheets("Sheet1").Range("B1").Value  
    oCust.sFirstName =  
Worksheets("Sheet1").Range("B2").Value  
    oCust.sLastName = Worksheets("Sheet1").Range("B3").Value  
    oCust.sStreet = Worksheets("Sheet1").Range("B4").Value  
    oCust.sCity = Worksheets("Sheet1").Range("B5").Value  
    oCust.sState = Worksheets("Sheet1").Range("B6").Value  
    oCust.sZipCode = Worksheets("Sheet1").Range("B7").Value  
  
    ' Save customer to table  
    If oCust.Save() Then  
        MsgBox "Customer information saved"  
    Else  
        MsgBox "Customer information NOT saved"  
    End If  
  
    Set oCust = Nothing  
  
Exit Sub  
  
SaveCust:  
    MsgBox oCust.sErrDesc & " (" & CStr(oCust.lErr) & ")"  
    Exit Sub  
End Sub
```

Listing 4-12: CustomerSave Procedure In Excel.

The only difference between the code shown above and the code from the Visual Basic sample is the method used to retrieve the object. In the Excel macro, you dim the variable as a generic Object. You can then use the CreateObject() function to create the Customer Data Service Routine object. Once you have this object, you can fill it with the data from the spreadsheet, and invoke the Save method.

2. Type OLESERV.EXE /REGSERVER and press OK.

Visual Basic adds start-up code that looks for this command line option. If it finds the code, Visual Basic creates a registry entry, it does not start the OLE server. The entry is contained under the HKEY_CLASSES_ROOT, and is listed under CustData.clsCustomer, the Project Name you entered under the Options menu.

To remove the registry entry for this version, run the .EXE file with the /UNREGSERVER command line option.

Now that you have made a real OLE server out of the Visual Basic program, you can use this OLE server from any application that supports OLE.

Now try to use the OLE server to insert a new customer from an Excel spreadsheet.

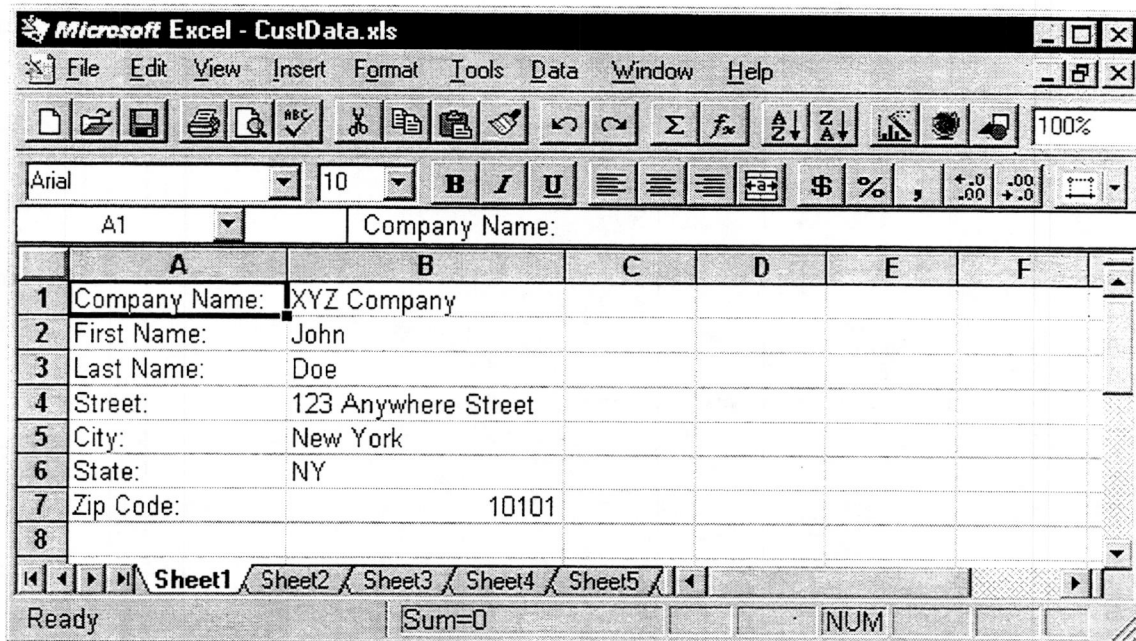


Figure 4-4: Data In Excel To Be Stored Using The OLE Server.

To begin, you can open CUSTDATA.XLS in Excel. You can enter any data into the information on Sheet1. Next go to the CustSave tab, and run the VBA code. If you now check the Microsoft Access database INVOICE.MDB, you can see that the new customer has been added. Here is the code you used in Excel:

15. Press F5 to run this OLE server. All that happens is the debug window is displayed, and the program continues to run.
16. Go back to Program Manager (or press the START button in Windows 95) and start another copy of Visual Basic 4.0.
17. Open the project OLETEST.VBP.
18. Double-click on the OLETEST.FRM.
19. Double-click on the Command button in the middle of the form.

You should see code that looks like the following:

```
Private Sub cmdCustInsert_Click()  
    Dim oCust As New clsCustomer  
  
    ' Set adding flag  
    oCust.AddFlag = True  
  
    ' Load customer data  
    oCust.CustomerID = 0  
    oCust.Company = "A New Company"  
    oCust.FirstName = "Dann"  
    oCust.LastName = "Jones"  
    oCust.Street = "123 Jonesy Street"  
    oCust.City = "Huntington Beach"  
    oCust.State = "CA"  
    oCust.Zip = "92555"  
  
    ' Save customer to table  
    If oCust.Save() Then  
        MsgBox "Customer information saved"  
    Else  
        MsgBox "Customer information NOT saved"  
    End If  
    Set oCust = Nothing  
End Sub
```

Listing 4-11: Customer Saving Using The OLE Server.

This is very similar to the code that you had in the Customer form in the previous example. The only difference is you have hard-coded the customer value to input.

The reason the object clsCustomer was recognized is because you set a reference to the "Customer System Data Routines." Look under Tools, References and you see a check box "Customer System Data Routines." This comes from the other Visual Basic program that is running.

If you stop the other instance of Visual Basic from running, then return to this instance, you should not see this reference anymore. What happens is that Visual Basic temporarily registers the OLE server so other applications can see it. When you end the program, Visual Basic cleans up the registry.

To Register An OLE Server After Making An .EXE File:

1. Select File, Run from the Program Manager.

Data Access OLE Server

You can use OLE servers either through OLE Automation or by embedding an object in a Visual Basic form and using Visual Basic as the OLE container. Using Visual Basic 4.0, you can create OLE servers that can be controlled through OLE Automation. You cannot create OLE servers that allow objects from the server to be embedded in another application.

So the question is, what kind of OLE server would you build using Visual Basic that can be useful to other programs? One of the best examples is a three-tier client/server application. This type of architecture is based on having your user interface as one .EXE, all of the business rules and data access routines as a separate .EXE or OLE server, and the database engine as the third level. Other uses of OLE servers are specialized applications like Excel, or a drawing package like Visio.

NOTE: You must have Visual Basic 4.0 Professional Edition or Enterprise Edition to run this sample.

The following steps help you create an OLE server from the Customer Class that you created earlier. If you think about how you created the Customer form, it is all user interface. The Customer Class is a separate .CLS file that can be used in any application, so it would be an ideal candidate for an OLE server.

Steps To Create An OLE Server:

1. Create a new project.
2. Create a .BAS file.
3. Add a Sub Main to this new .BAS file.
4. Select File, Add File, and add the file O_CUST.CLS or J_CUST.CLS.
5. Double-click on clsCustomer class object.
6. Press F4 and set the Instancing property to 2-Creatable MultiUse.
7. Set the Public property to True.
8. Select File, Add File, and add the file DATA.BAS.
9. Select File, Add File, and add the file WINAPI.BAS.
10. Select Tools, References, and check the Microsoft DAO 2.5/3.0 Compatibility Layer.
11. Select Tools, Options, and click on the Project tab.
12. Set the Project Name to "CustData".
13. Click the OLE server under the Startup Mode frame.
14. Set the Application Description to "Customer Data Routines".

Running From Excel

It is time to test the OLE server from Microsoft Excel. To accomplish this, follow the steps below:

Master Steps:

1. Start your OLE server in Visual Basic Design Mode.
2. Start up Microsoft Excel 5.
3. Type in the code below.

```
Sub UpdateSales()  
    Dim oSales As Object  
  
    Set oSales = CreateObject("Sales.QtrSales")  
  
    oSales.Quarter = "1"  
    oSales.Year = "1996"  
    oSales.Sales = 250000  
    If oSales.WriteSales() Then  
        MsgBox "Sales successfully written"  
    Else  
        MsgBox "Sales NOT written"  
    End If  
End Sub
```

Listing 4-10: UpdateSales() In Microsoft Excel 5.0.

You will notice a slight difference in the code used in Microsoft Excel compared to what you used in Visual Basic. The above code would have worked within Visual Basic as well.

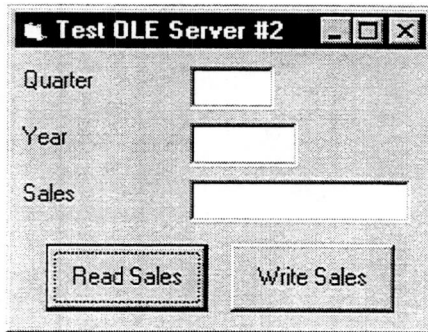


Figure 4-3: OLE Test Example

This project will allow you to test the OLE server by invoking the `GetSales()` and `WriteSales()` methods. Take a look at the code to read the sales:

```
Private Sub cmdRead_Click()
    Dim oSales As New Sales.QtrSales

    Screen.MousePointer = vbHourglass
    If oSales.GetSales() Then
        txtQtr = oSales.Quarter
        txtYear = oSales.Year
        txtSales = oSales.Sales
    Else
        MsgBox "Sales.NOT Retrieved"
    End If
    Screen.MousePointer = vbDefault
End Sub
```

Listing 4-8: Reading Sales Information.

If you click on the Read Sales command button, the `GetSales` method is called for the `oSales` object. This routine will fetch the data from the Database specified in the method, and fill in the appropriate properties.

Now call the `WriteSales()` method:

```
Private Sub cmdWrite_Click()
    Dim oSales As Object

    Set oSales = New Sales.QtrSales

    oSales.Quarter = txtQtr
    oSales.Year = txtYear
    oSales.Sales = txtSales

    If oSales.WriteSales() Then
        MsgBox "Sales figures stored"
    Else
        MsgBox "Error writing sales"
    End If
End Sub
```

Listing 4-9: Writing Sales To tblSales.

```
Public Function WriteSales() As Boolean
    Dim db As Database
    Dim sSQL As String

    On Error GoTo WriteSales_EH

    Set db = DBEngine(0).OpenDatabase(DB_NAME)

    sSQL = "INSERT INTO tblSales ("
    sSQL = sSQL & "sQtr, sYear, lSales ) "
    sSQL = sSQL & "VALUES ( "
    sSQL = sSQL & pstrQtr & ", "
    sSQL = sSQL & pstrYear & ", "
    sSQL = sSQL & plngSales & ") "

    db.Execute sSQL

    db.Close

    WriteSales = True

    Exit Function

WriteSales_EH:
    WriteSales = False
    Exit Function
End Function
```

Listing 4-7: WriteSales Method.

Master Steps:

1. Select Tools, Options from the menu.
2. Select the Project Tab.
3. Set the Project Name to Sales. This will be used to qualify the Class that will be used from this project in any application other than Visual Basic.
4. Set the Application Name to "Populate Sales Table". This will be the name that will show up in the References section of any OLE application that needs to use this Class.
5. Also set the StartMode to OLE server. This will allow you to test this application from the Visual Basic design environment.
6. Once this is complete, click on the OK button.
7. Now run this project.
8. The application will appear to be running but nothing will be happening.

Testing The OLE Server

Open another instance of Visual Basic and load the sample project below:

```
Public Function GetSales() As Boolean
    Dim db As Database
    Dim snpData As Recordset
    Dim sSQL As String

    On Error GoTo GetSales_EH

    Set db = DBEngine(0).OpenDatabase(DB_NAME)
    sSQL = "SELECT * FROM tblSales"
    Set snpData = db.OpenRecordset(sSQL, dbOpenSnapshot)

    If snpData.EOF Then
        GetSales = False
    Else
        pstrQtr = snpData!sQtr
        pstrYear = snpData!sYear
        plngSales = snpData!lSales
        GetSales = True
    End If
    snpData.Close
    db.Close

    Exit Function

GetSales_EH:
    GetSales = False
    Exit Function
End Function
```

Listing 4-5: GetSales() Method.

Write Properties

We also need to allow the ability to write to each of these properties. Add the following Property Let procedures to this .CLS file:

```
Property Let Quarter(ByVal sQuarter As String)
    pstrQtr = sQuarter
End Property
Property Let Sales(ByVal lSalesIn As Long)
    plngSales = lSalesIn
End Property
Property Let Year(ByVal sYearIn As String)
    pstrYear = sYearIn
End Property
```

Listing 4-6: Property Let Procedures.

You also need a Public method that will allow you to write the information contained within these properties out to a Microsoft Access .MDB file:

Another OLE Server

As another example, create an OLE server that will allow any OLE client application to enter data into an Access MDB table.

Master Steps:

1. Create a new project.
2. Remove the default form.
3. Insert a .BAS file.
4. Add a Sub Main() to the .BAS file.
5. Insert a .CLS file.
6. Press F4 to bring up the properties window.
7. Set the Name property to QtrSales.
8. Set the Instancing property to 2 - Creatable Multi-use.
9. Set the Public property to True.
10. Add the following Private properties to the .CLS file in the (General) (Declarations) area:
Option Explicit

```
Private pstrQtr As String
Private pstrYear As String
Private plngSales As Long
```

Read Properties

You need to be able to retrieve these properties from your client application. Add the following Property Get procedures to your class:

```
Property Get Quarter() As String
    Quarter = pstrQtr
End Property
Property Get Sales() As Long
    Sales = plngSales
End Property
Property Get Year() As String
    Year = pstrYear
End Property
```

Listing 4-4: Property Get Procedures.

Read Method

Now create a very simple method to retrieve the first row from tblSales table, and fill in the properties with the values in this row.

Master Steps:

1. Select Tools, References from the menu.
2. Find the Available Reference named DateReturn and select it. This is our OLE server that is running in the other instance of Visual Basic.
3. Press the OK button.
4. Run the project and click on the command button. If everything went OK, you should see a Message Box that looks like the following (except it will have a different date and time):

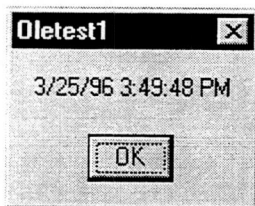


Figure 4-2: Message Box From An OLE Server.

Congratulations! You have created your first OLE server in Visual Basic.

When you create an instance of this class, the `dtDate` property will be defaulted to the current date and time.

Creating The Server

Now you just need to fill in a couple of other items to create an OLE server.

Master Steps:

1. Select Tools, Options from the menu.
2. Select the Project Tab.
3. Set the Project Name to `DateReturn`. This will be used to qualify the Class that will be used from this project in any application other than Visual Basic.
4. Set the Application Name to "Return a Date". This will be the name that will show up in the References section of any OLE application that needs to use this class.
5. Also set the `StartMode` to OLE server. This will allow you to test this application from the Visual Basic design environment.
6. Once this is complete, click on the OK button.
7. Now run this project.
8. The application will appear to be running but nothing will be happening.

Testing The Server

Open another instance of Visual Basic and open the test project for this application.

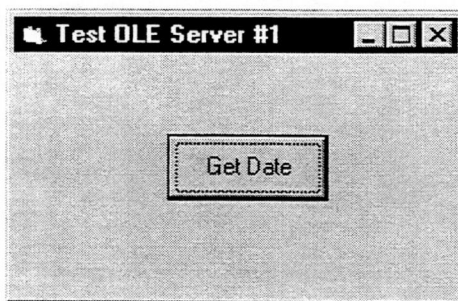


Figure 4-1: OLE Example

In the click event of the command button, use the following code to retrieve the date from the OLE server.

```
Private Sub cmdDate_Click()  
    Dim oDate As New DateReturn.ReturnDate  
  
    MsgBox oDate.DateStarted  
End Sub
```

Listing 4-3: Retrieving A Date From The OLE Server.

Creating A Simple OLE Server

A simple example of an OLE server is a Class that simply returns one value; a date. This date will be the date and time the server was started. To make this into an OLE server, you need to perform the following steps:

Master Steps:

1. Create a new project.
2. Remove the default form.
3. Insert a .BAS file.
4. Add a Sub Main() to the .BAS file.
5. Insert a .CLS file.
6. Press F4 to bring up the properties window.
7. Set the Name property to ReturnDate.
8. Set the Instancing property to 2 - Creatable Multi-use.
9. Set the Public property to True.
10. Add one Private property to the .CLS file in the (General) (Declarations) area.
Option Explicit

```
Private dtDate As Date
```

Read Properties

After creating the Private property you need to allow the ability to read the property. Add a Property Get procedure to this .CLS file:

```
Property Get DateStarted() As Date  
    DateStarted = dtDate  
End Property
```

Listing 4-1: Property Get Procedure.

The above code will return the value of the Private property dtDate.

Class Initialization

Now you need to initialize this date to the current date and time. In the Class_Initialize event, add the following code:

```
Private Sub Class_Initialize()  
    dtDate = Now  
End Sub
```

Listing 4-2: Class_Initialize() Event.

In an earlier sample application, you programmatically checked some simple business rules to make sure the company name is filled in, the invoice description is not blank, that there is a valid billing date filled in, etc. These are just some simple examples of business rules. All of these are perfect candidates for encapsulation within an OLE server.

Business Rules

An OLE server in a three-tier client/server architecture helps you encapsulate business rules. Business rules have typically been embedded either in the application code, or in stored procedures on a database server.

Traditional Approach

Suppose that you have an application with your business rules built into the application. If you choose to build a new application that needs to use those rules, you have to either duplicate the code or use the same library of routines. If you duplicate the code, whenever a change has to be made, you must change both applications. Additionally, you need to recompile the applications and redistribute them to all users. The advantage to having the business rules in the application is that it provides immediate feedback to the user when a rule is violated. If you have to send the SQL statement to the back end and then return an error message, this process takes time and requires network and server resources.

Putting rules on the back end, in stored procedures for example, is great because if a rule changes, you only need to change the rule in one place. Then all applications can take advantage of the change immediately without any coding changes. However, many times you are limited to SQL to implement rules. Some rules can get quite complicated, and as such do not lend themselves well to stored procedures.

OLE Server Approach

Why not use another software layer, such as an OLE server, to implement business rules? You get the benefits of one central repository for all the business rules, you can potentially cut down the time it takes to process a rule, and you can use a robust programming language (Visual Basic) to implement complicated rule processing.

Rules To Encapsulate

How do you determine which rules to put into an OLE server? Begin by taking rules that are buried in the application and moving them into class objects. These objects perform validation checking based on the rules or conditions that you establish in an object's properties. If everything looks OK, the object would submit the SQL statement to the back end for processing. If there were errors, the object would return an error message to the application that you would communicate to the user.

The big question is whether to get rid of your stored procedures from your back-end server and move them into the OLE server. **ABSOLUTELY NOT!** It is vital that the database engine be able to protect its data from rule violations. The data needs protection from users who bypass the OLE server using Microsoft Access or MS Query and attempt to manipulate the data directly in the database.

Examples Of Rules

Three-tier Client/Server Architecture

You have learned how to create objects and how to break applications into separate pieces. These are the steps in moving towards a three-tier client/server architecture.

Tier 1

The first tier in the architecture is the user interface tier. This tier is responsible for displaying information to the user and retrieving information from the user. It is also responsible for calling the 2nd tier and passing all the information it has retrieved. The first tier normally is created with a front end tool such as Visual Basic 4.0, Microsoft Access, Visual FoxPro, or another application generation tool.

Tier 2

The second tier is responsible for validating data and submitting the data to the back-end database for storage. It can also be responsible for retrieving data and error messages from the back-end database and communicating the information back to the 1st tier. The second tier normally is an OLE server created with Visual Basic 4.0 or maybe with C++.

Tier 3

The third tier is responsible for managing the data. This includes inserting, updating, and deleting rows; maintaining referential integrity; rule checking, and any other data-centric functions normally associated with a relational database system. Third tiers are any back-end databases such as Microsoft SQL Server, Oracle, Sybase, etc.

NOTE: The third tier does not include ISAM databases such as Microsoft Access, Xbase, Paradox, Btrieve, etc. Since there is no server handling requests, everything is still processed by the client application.

OLE Automation Servers

Probably the most exciting technology in Visual Basic 4.0 is the ability to create OLE servers. By building class modules and exposing properties and methods, you can create an application that has the ability to be called from any OLE client. You are allowed to create both in-process (DLLs) and out-of-process (EXE) OLE servers. Using this technology, you can easily build a three-tier client/server architecture.

Another feature of the OLE server is the ability to run the OLE server on any workstation in your network. This means you can put your business rules and data access routines on one very powerful machine, and all of your users can use the CPU of this other machine to perform validation and data manipulation routines.

Introduction

With Visual Basic 4.0, you can now create OLE Automation servers! OLE Automation servers help fill in another piece in the client/server puzzle. Many developers have heard the buzzword *three-tier architecture*, but few people can actually define it. Many developers will be confused at first about exactly how to implement it and what three-tier architecture really is. You will learn one methodology for designing a three-tier client/server application.

Objectives

In this section you will:

- Learn the advantages of three-tier client/server architecture.
- Learn how you might distribute your application's business rules.
- Walk through the steps for creating an OLE server.

Creating OLE Servers

- Ease of maintenance.
- Separation of forms and data access routines.
- Ability to change the underlying data access method.
- Allows multiple applications to use a common data access method.
- You can create OLE servers easily out of the data access methods.

NOTE: Visual Basic 4.0 is moving towards object-oriented programming. Although Visual Basic 4.0 does not support inheritance, you can take advantage of building objects that perform many encapsulated functions. This is what OOP is mostly about anyway, hiding the implementation details of an object.


```

Else
    Call FormClear
    mboolAddRecords = True
End If
moInvoice.ResultClose

Call StatusRestore(strOldMsg)
End Sub

```

Listing 3-17: lstInvoiceLoad() Routine.

Another example is the Delete routine in the Customer and Invoice forms. Remember that two different objects were declared so the Delete method is applied to each object.

```

moCust.Delete()
moInvoice.Delete()

```

While the method is the same, the code to perform the delete is completely different. It all depends on what the object is that prefaces the Delete() method:

```

Public Function Delete() As Integer
    Dim strSQL As String

    strSQL = "DELETE FROM tblCustomers "
    strSQL = strSQL & "WHERE lCust_id = " & lngCustID
    If SQLSubmit(strSQL) Then
        Delete = True
    Else
        Delete = False
    End If
End Function

Public Function Delete() As Integer
    Dim strSQL As String

    strSQL = "DELETE FROM tblInvoice "
    strSQL = strSQL & "WHERE lInvoice_id = " & lInvoiceID
    If SQLSubmit(strSQL) Then
        Delete = True
    Else
        Delete = False
    End If
End Function

```

Listing 3-18: Function Delete().

As you can see, the two functions are different in that the SQL statement either uses the tblInvoice or tblCustomers table.

Benefits Of Data Access Wrapping

Our whole framework is designed around separating the user interface routines from the data access routines. While this may not be very efficient from a performance standpoint, there are many benefits to this design, including:

- Polymorphism allows one standard set of methods for each business object.

```
' Retrieve first customer record
boolPerform = moCust.GetFirstRow()
If boolPerform Then
    lstNames.Clear

    Do While boolPerform
        lstNames.AddItem moCust.strCompany
        lstNames.ItemData(lstNames.NewIndex) = _
            moCust.lngCustID
        ' Retrieve next record
        boolPerform = moCust.GetNextRow()
    Loop

    ' Trigger a call to FormShow if data found
    If lstNames.ListCount > 0 Then
        lstNames.ListIndex = 0
    End If
Else
    Call FormClear
    mboolAddRecords = True
End If
moCust.ResultClose

Call StatusRestore(strOldMsg)
End Sub
Listing 3-16: lstCustomerLoad() Routine.
```

Now look at the similarities in the `lstInvoiceLoad()` routine. The names of the methods to load the data are exactly the same.

```
Private Sub lstInvoiceLoad()
    Dim strOldMsg As String
    Dim boolPerform As Integer

    strOldMsg = StatusProcess("Please wait while loading invoice list")

    moInvoice.lngCustID = moCust.lngCustID
    ' Retrieve first invoice record
    boolPerform = moInvoice.GetFirstRow()
    If boolPerform Then
        lstInvoice.Clear

        Do While boolPerform
            lstInvoice.AddItem moInvoice.dtBillingDate & _
                Chr$(9) & moInvoice.sInvoiceDesc
            lstInvoice.ItemData(lstInvoice.NewIndex) = _
                CLng(moInvoice.lInvoiceID)
            ' Retrieve next invoice record
            boolPerform = moInvoice.GetNextRow()
        Loop

        ' Trigger a call to FormShow if data found
        If lstInvoice.ListCount > 0 Then
            lstInvoice.ListIndex = 0
        End If
    End If
End Sub
```

The actual function that loads the data from the Snapshot into the properties is called `SetProperties()`.

```
Private Sub SetProperties()
    lngCustId = Field2Long(msnpData!lCust_id)
    strCompany = Field2Str(msnpData!szCompany_nm)
    strLastName = Field2Str(msnpData!szLast_nm)
    strFirstName = Field2Str(msnpData!szFirst_nm)
    strStreet = Field2Str(msnpData!szStreet_ad)
    strCity = Field2Str(msnpData!szCity_nm)
    strState = Field2Str(msnpData!strState_cd)
    strZipCode = Field2Str(msnpData!sZip_cd)
    boolActive = Field2Int(msnpData!iActive_cd)

    intConcur = IIf(IsNull(msnpData!iConcurrency_id), _
        -1, msnpData!iConcurrency_id)
End Sub
```

Listing 3-14: `SetProperties()` Method.

Going back to the routine that is loading the list box, `lstCustomerLoad()`, this routine only loads the data from the properties of the customer object. It never directly accesses the Snapshot object at all.

Polymorphism

One of the features of using objects is the concept of polymorphism. This concept gives the developer the freedom to ignore the code behind an object's methods and concentrate on programming with the object. For example, most Visual Basic controls have the method *move*. As a developer, you can use the line

```
MyControl.Move 0,0,0,0
```

and not concern yourself with how the object will move itself.

In this situation, the interface is the same for each form in the project. If you compare the Customer form with the Invoice form, you will notice many similarities.

First you Dim two module level variables in each form:

```
' Customer form
Dim moCust As New clsCust

' Invoice form
Dim moInvoice As New clsInvoice
```

Listing 3-15: Declaring Object Variables

Now look at the `lstCustomerLoad()` routine:

```
Private Sub lstCustLoad()
    Dim strOldMsg As String
    Dim boolPerform As Integer

    strOldMsg = StatusProcess("Please Wait While " & _
        "Loading Customer List")
```

The first method called, `GetFirstRow()`, creates a Snapshot of customer information and then loads the first row into the properties of the customer object.

```
Public Function GetFirstRow() As Integer
    Dim strSQL As String

    On Error GoTo GetFirstRow_EH

    GetFirstRow = False

    Call ResultClose

    mboolSnapActive = True
    strSQL = SQL(False)
    Set msnpData = goLogin.db.OpenRecordset(strSQL, _
        dbOpenSnapshot)

    If Not msnpData.EOF Then
        GetFirstRow = GetNextRow()
    End If

    Exit Function

GetFirstRow_EH:
    Dim intAction As Integer
    intAction = ErrHandler(Err)
    ...
    ...
End Function
```

Listing 3-12: `GetFirstRow()` Method.

The `GetNextRow()` method is responsible for calling the routine to load the properties and move to the next row in the result set.

```
Public Function GetNextRow() As Integer
    GetNextRow = False

    On Error GoTo GetNextRow_EH

    If mboolSnapActive Then
        If Not msnpData.EOF Then
            Call SetProperty
            msnpData.MoveNext
            GetNextRow = True
        End If
    End If

    Exit Function

GetNextRow_EH:
    Dim intAction As Integer
    intAction = ErrHandler(Err)
    ...
    ...
End Function
```

Listing 3-13: `GetNextRow()` Method.

- Save
- SetPropertyies
- SQL

For example, you need to load the customer list box. The list box is a Visual Basic control. You do not want to create a method in the object that uses any controls at all. Instead, you want to use properties or variables that are useable from any application.

In the customer Form, you can write code that calls the methods in the object and loads the list box. This way the object does not need to connect to any particular control that is being loaded, it only loads up its properties.

```
Private Sub lstCustLoad()
    Dim strOldMsg As String
    Dim boolPerform As Integer

    strOldMsg = StatusProcess("Please Wait While " & _
        "Loading Customer List")

    ' Retrieve first customer record
    boolPerform = moCust.GetFirstRow()
    If boolPerform Then
        lstNames.Clear

        Do While boolPerform
            lstNames.AddItem moCust.strCompany
            lstNames.ItemData(lstNames.NewIndex) = _
                moCust.lngCustId
            ' Retrieve next record
            boolPerform = moCust.GetNextRow()
        Loop

        ' Trigger a call to FormShow if data found
        If lstNames.ListCount > 0 Then
            lstNames.ListIndex = 0
        End If
    Else
        Call FormClear
        mboolAddRecords = True
    End If
    moCust.ResultClose

    Call StatusRestore(strOldMsg)
End Sub
```

Listing 3-11: Loading A Customer List Box.

Prior to calling the above code to load the customer list box, declare a module level variable in the form to hold the customer object:

```
Dim moCust As New clsCust
```

When the form is displayed, this object variable is created. Now, load the data by calling each of the methods in bold in the "Loading A Customer List Box" listing above.

Using Classes For Data Access Methods

You now have a feel for using class objects to handle things like initializing and opening databases. Here is an example of using objects for loading data into forms.

Typically Visual Basic programmers write code to handle keystrokes and other events in the forms themselves, along with code to extract the data from a table and display it on that form. Another approach is to create objects for all database handling routines, then use the objects within the forms themselves.

Customer Class

To begin, build a customer object that mimics the structure of the tblCustomers table located in the database. This structure allows you to use object methods to read, save, and modify data in the customer table. Since the object is separate from the form, you can use the object in other places without having to write additional code to perform the table manipulation. Following are the properties for this object:

```
Public lngCustId As Long
Public strCompany As String
Public strFirstName As String
Public strLastName As String
Public boolActive As Integer
Public strStreet As String
Public strCity As String
Public strState As String
Public strZipCode As String
Public intConcur As Integer

Public boolAdding As Boolean

Private msnpData As Recordset
Private mboolSnapActive As Boolean
```

Listing 3-10: Properties For The Customer Class.

The properties listed above all match the columns in the tblCustomers table. There are also some additional properties that you can use to manipulate recordsets within the object. Next, you need to build some methods that read data into these properties or write the information from the properties out to the table. The following are some of the methods in the customer object:

- GetFirstRow
- GetNextRow
- GetRow
- ResultClose
- Delete

In this DataOpen() method, you use the SQLConnect() function to open a connection to the DSN specified in the CUST.INI file or in the system registry for the application.

```
        Set db = ws.OpenDatabase("", False, False, _
            strConnect)

        Screen.MousePointer = DEFAULT
    Else
        Set db = ws.OpenDatabase(strDatabase)
    End If
End If
Call StatusRestore(strOldMsg)

boolValidLogon = False

Exit Function

DataOpen_EH:
    Dim intErr As Integer
    intErr = ErrHandler(Err)
    DataOpen = False
    boolValidLogon = False

    Exit Function
End Function
Listing 3-8: DataOpen() Method For Jet.
```

In the DataOpen() for Jet listing, you used the Workspace object that was initialized with the DataInit() method to open the database object. Now look at the code to establish a connection using the ODBC method:

```
Public Function DataOpen() As Integer
    Dim strResult As String
    Dim intSize As Integer
    Dim strConnect As String
    Dim strOldMsg As String
    Dim intStatus As Integer

    DataOpen = False

    If boolValidLogon Then
        strOldMsg = StatusProcess("Attempting To Logon " & _
            "To Database...")

        intStatus = SQLConnect lngdbc, strDSN, Len(strDSN), _
            strLoginID, Len(strLoginID), strPassword, _
            Len(strPassword))
        If intStatus = SQL_ERROR Then
            DataOpen = False
        Else
            DataOpen = True
        End If

        Call StatusRestore(strOldMsg)
    End If
End Function
Listing 3-9: DataOpen() Using The ODBC API.
```



```
        ' Free the environment
        intStatus = SQLFreeEnv(lngEnv)
        If intStatus = SQL_ERROR Then
            Call ODBCErrorMsg(lngEnv, lngdbc, 0&,
                "Could not free memory from ODBC driver", _
                "ODBCInit()")
        End If
        boolValidLogon = False
    End If
End If

DataInit = boolValidLogon
End Function
```

Listing 3-7: DataInit() Procedure For ODBC API.

As you can see in the two versions of code, the methodology for initializing the database engines is quite different. The Sub Main() routine, however, did not have to change at all. You always call DataInit() and this method hides the implementation details from you. This is one of the great benefits of object-oriented programming. All you need to do is change the class; the main body of code does not change.

DataOpen()

After initializing the database engine, open the database. You can create a Public Method called DataOpen() to perform the opening of the database. This routine again differs depending on the engine used for database access. Take a look at both the Jet and ODBC API versions of this DataOpen() method.

```
Public Function DataOpen() As Integer
    Dim strOldMsg As String
    Dim strConnect As String

    On Error GoTo DataOpen_EH

    strOldMsg = StatusProcess("Opening Database...")
    DataOpen = True
    If strDatabase = "" Then
        MsgBox "No Database Name Filled In, " & _
            "Please Check the File: '" & gstrIniName _
            & "'"
        DataOpen = False
    Else
        If goLogin.boolServer Then
            Screen.MousePointer = HOURLGLASS

            strConnect = "ODBC;"
            strConnect = strConnect & "DSN=" & strDSN
            strConnect = strConnect & ";UID=" & strLoginID
            strConnect = strConnect & ";PWD=" & strPassword
            strConnect = strConnect & ";DATABASE=" & _
                strDatabase
            strConnect = strConnect & ";APP=" & APP_CODE
```

DataInit()

The method DataInit() changes based on the database engine being used. When using the Jet engine, you can use DataInit() to setup your workspace for the particular database to be opened. If you are using ODBC, DataInit() can allocate memory for the environment and the connection handle. Below is an example from the Jet engine:

```
Public Function DataInit() As Integer
    On Error GoTo DataInit_EH
    Set ws = DBEngine.Workspaces(gintWorkSpace)
    gintWorkSpace = gintWorkSpace + 1
    DataInit = True
Exit Function

DataInit_EH:
    DataInit = False
    Exit Function
End Function
```

Listing 3-6: DataInit() For Jet.

If you use the ODBC API, you need to modify this class to use two Long integers instead of the Database and Workspace objects:

```
Public lngEnv As Long ' Environment handle for ODBC API
Public lngdbc As Long ' Database connection handle for
                    ' ODBC API
```

Next, change the DataInit() function to use the ODBC API to allocate memory for an environment handle and a database connection handle.

```
Public Function DataInit() As Integer
    Dim intStatus As Integer

    boolValidLogon = True

    ' This ODBC call checks to see if the libraries are
    ' present and if we can load them into memory
    ' and establish a memory area for them to work in.
    intStatus = SQLAllocEnv(lngEnv)
    If intStatus = SQL_ERROR Then
        Call ODBCErrorMsg(lngEnv, lngdbc, 0&, _
            "Unable to initialize ODBC API drivers", _
            "ODBCInit()")
        boolValidLogon = False
    End If

    If boolValidLogon Then
        ' This call establishes a memory area where we
        ' can put login information that will be used to
        ' login to our data source. The area for the
        ' login information is contained in the lngdbc
        ' variable.
        intStatus = SQLAllocConnect(lngEnv, lngdbc)
        If intStatus = SQL_ERROR Then
            Call ODBCErrorMsg(lngEnv, lngdbc, 0&, _
                "Could not allocate memory for " & _
                "connection handle", "ODBCInit()")
```

Database/Login Class

Visual Basic 4.0 lets you create classes that you can use to group information such as Database objects, DSN names, Login IDs, passwords, etc. This example will show you how to create a Login class with the following properties:

```
Public strDatabase As String      ' Database name
Public strDSN As String          ' DSN for ODBC connections

Public ws As Workspace           ' Workspace object
Public db As Database            ' Database object

Public lngUserID As Long         ' User ID from user table
Public lngUserGroup As Long      ' Group ID from user table
Public strLoginID As String      ' User login ID
Public strPassword As String     ' User password

Public strUserName As String     ' Full user name for display
Public boolServer As Boolean     ' Connection for SQL Server?
Public boolValidLogon As Boolean ' Valid logon occurred?
```

Listing 3-4: Properties for Jet Login Class.

NOTE: The listing above uses all Public variables, although normally you would create Private variables and use Property Get/Let functions to set the properties.

You can create a public object called *goLogin* to login to the database. There are three main methods that make up this class:

- DataInit()
- DataOpen()
- DataClose()

The three methods are used to initialize the database engine, open the database engine, and close the database engine respectively. To use this class, you need to first create a global variable:

```
Global goLogin as New clsLogin
```

The above code creates a new instance of that class and assigns the object to the variable named *goLogin*. Next, fill in the login ID and password from the login form, and call the DataInit() and DataOpen() methods:

```
goLogin.strLoginID = Trim$(txtName.Text)
goLogin.strPassword = txtPassword.Text

boolValid = goLogin.DataInit()
If boolValid Then
    boolValid = goLogin.DataOpen()
End If
```

Listing 3-5: Logging Into The Database.

Login Form

The first form the user sees after the main form is a login form. For all SQL databases, you must login with a user name and password. You can create your own login form instead of relying on the ODBC driver to display its dialog box. This helps maintain consistency as users move from one database system to another.

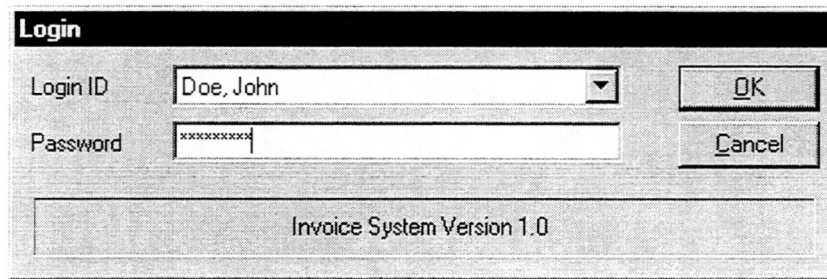


Figure 3-3: Login Form.

After the user logs into the application, take the login information and open the data source using the login ID and password. To accomplish this, you can create a Database/Login class.

When you retrieve information from the system registry, the information can be contained under the following entries:

HKEY_USERS on Local Machine
 Software
 VB and VBA Program Settings
 <YourAppName>
 <YourSections>

The function above checks to see if the entries exist in the system registry. If not, you need to call a routine to create defaults:

```
Sub AppCreateEnv()  
    ' Set default login ID  
    SaveSetting gstrIniName, "Logon", "UserName", _  
        "UserName"  
  
    ' Set location where DSN is located  
    SaveSetting gstrIniName, "DataSource", "DataLoc", _  
        "Local"  
  
    ' Set data source name  
    SaveSetting gstrIniName, "Local", "DSN", " "  
  
    ' Set default database name  
    SaveSetting gstrIniName, "Local", "DBName", _  
        App.Path & "\" & "invoice.mdb"  
  
    ' Set server flag  
    SaveSetting gstrIniName, "Local", "Server", "0"  
End Sub
```

Listing 3-3: AppCreateEnv().

```
If goLogin.boolValidLogon Then
    ' Display the login form
    frmLogin.Show MODAL
End If

' Did the user login correctly?
If Not goLogin.boolValidLogon Then
    ' If not, unload the main form
    Unload frmMain
Else
    ' Display the customer form
    frmCust.Show
End If

Screen.MousePointer = DEFAULT
End Sub
Listing 3-1: Sub Main().
```

The first thing the Main routine does is display the copyright form. This is where you can put your company logo or another splash screen. The user needs something to look at while you load global variables with information from your .INI file or from the system registry. The routine used to retrieve information from the .INI file or from the system registry looks like the following:

```
Sub AppGetEnv()
    Dim strDataLoc As String
    Dim vntAll As Variant

    ' Check for existence of INI or System Registry
    vntAll = GetAllSettings(gstrIniName, "DataSource")
    If IsEmpty(vntAll) Then
        Call AppCreateEnv
    End If

    ' Retrieve default logon name
    goLogin.strLoginID = GetSetting(gstrIniName, "Logon", _
        "UserName", "")

    ' Retrieve location where DSN is located
    strDataLoc = GetSetting(gstrIniName, "DataSource", _
        "DataLoc", "")

    ' Retrieve data source name
    goLogin.strDSN = GetSetting(gstrIniName, strDataLoc, _
        "DSN", "")

    ' Retrieve database name
    goLogin.strDatabase = GetSetting(gstrIniName, _
        strDataLoc, "DBName", "")

    ' Get server flag
    gboolServer = CInt(Val(GetSetting(gstrIniName, _
        strDataLoc, "DBServer", "0")))
End Sub
Listing 3-2: AppGetEnv() Procedure.
```

Sub Main()

Most professional Visual Basic programmers start their applications with Sub Main() instead of a startup form. Sub Main() gives you more flexibility to initialize variables, read information from the system registry or an .INI file, display a copyright form, etc.

Below is sample code for our Sub Main() example:

```
Sub Main()  
    Dim strTime As String  
  
    Screen.MousePointer = HOURGLASS  
  
    ' Set display start time  
    strTime = Now  
    ' Display copyright form  
    frmCopyright.Show  
    frmCopyright.Refresh  
  
    ' Retrieve application globals from an .INI file  
    ' or from the System Registry  
    #If Win16 Then  
        gstrIniName = App.Path & "\" & App.EXENAME & ".INI"  
    #Else  
        gstrIniName = App.EXENAME  
    #End If  
    Call AppGetEnv  
  
    ' If you are connecting to a SQL database,  
    ' don't open a connection until you've  
    ' asked the user for login ID and password  
    If gboolServer Then  
        goLogin.boolValidLogon = True  
    Else  
        goLogin.boolValidLogon = goLogin.DataInit()  
        If goLogin.boolValidLogon Then  
            goLogin.boolValidLogon = goLogin.DataOpen()  
        End If  
    End If  
  
    If goLogin.boolValidLogon Then  
        ' Preload main form  
        Load frmMain  
  
        ' Display copyright form for 2.5 seconds  
        Do Until DateDiff("s", strTime, Now) > 2.5  
            DoEvents  
        Loop  
        ' Now unload the copyright form  
        Unload frmCopyright  
  
        ' Display main form  
        frmMain.Show  
    End If
```

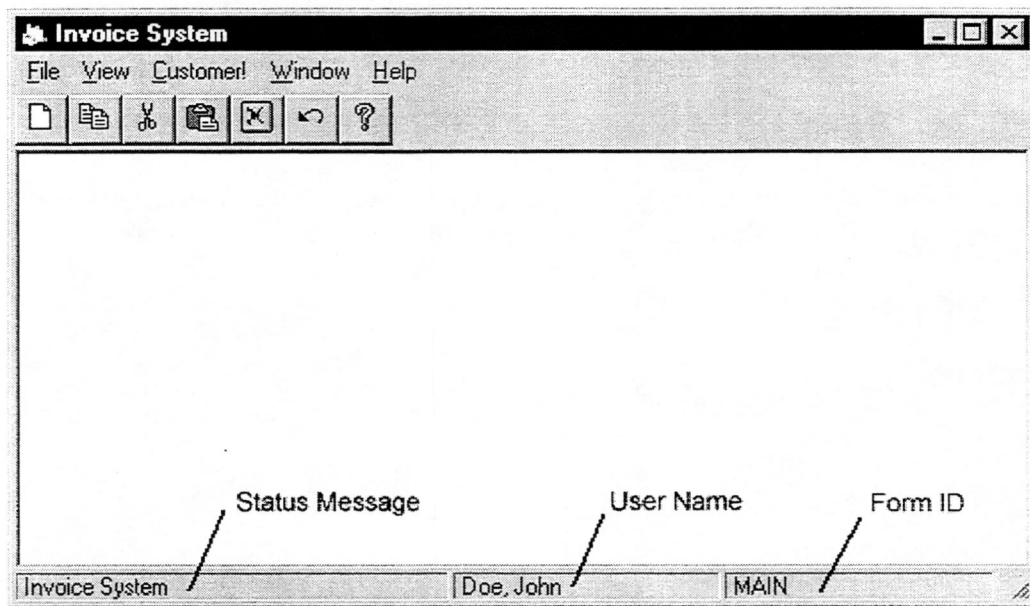


Figure 3-2: Status Bar Message Areas.

Multiple Document Interface

Our business application is based on the Multiple Document Interface (MDI) model. Most business applications are designed around this paradigm, as it encapsulates the whole application within one common window.

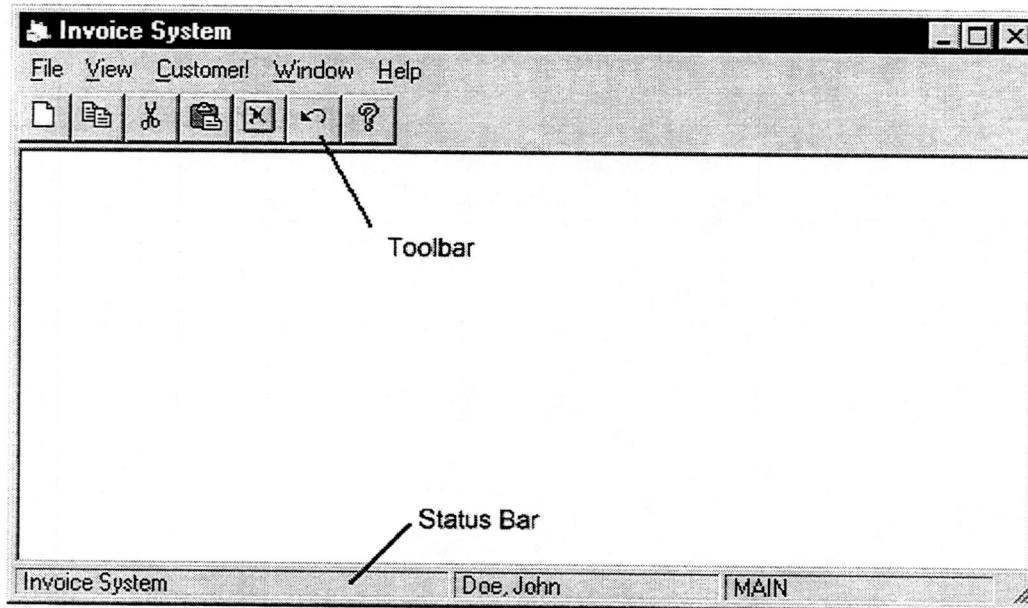


Figure 3-1: Toolbars And Status Bars On A Main MDI Form.

Status Bars And Toolbars

Most MDI windows contain a Status Bar and Toolbar area. You can use them in different ways, and will do so in the business application developed in this class. The Status Bar area contains one large area for status messages, one area for the user who is logged into the application, and one more area for a form identification code.

Why A Framework?

Visual Basic 4.0 is a robust environment for creating business applications. Like any Windows development system, it can be difficult to determine where to put the code to make the application run the best and still handle multiple database platforms. In this section, you'll see a typical Visual Basic application that provides a starting point for your own Visual Basic programming.

NOTE: Although this is one solution to how an application framework should be designed, it is certainly not the only way. Please use the information given here to assist you in developing your own application framework that best fits your individual needs.

Introduction

This chapter provides you with a sample application that show you one method for designing Visual Basic 4.0 applications. This method can easily be converted to use any data access methodology, or converted to a three-tier client/server architecture. You will learn how this framework was built, and why.

Objectives

In this section you will:

- Learn to create an application based on the Multiple Document Interface (MDI).
- Create a Status Bar for your application.
- Learn to use Sub Main().
- Learn why to use a Login form.
- Explain the advantages of a Database/Login Class.
- Use classes for data access.

Application Framework

Comparison Between Jet DAO And RDO

There are several similarities and also several differences between Jet and the RDO access objects. Below is a table outlining the objects that are equivalent between the two methodologies, and those that have no equivalent.

Jet	RDO
DBEngine	rdoEngine
Workspace	rdoEnvironment
Database	rdoConnection
Recordset	rdoResultset
Dynaset	KeySet rdoResultset
Snapshot	Static rdoResultset
Not Available	Dynamic rdoResultset
Table	Not implemented
Not Available	Forward-only rdoResultset
forward-only-scrolling snapshot	Not Available
Error	rdoError
Field	rdoColumn
QueryDef	rdoPreparedStatement
TableDef	rdoTable
Parameter	rdoParameter
record	row
field	column

Table 2-7: Comparison Between Jet And The RDO Models.

Advantages/Disadvantages Of RDO

The main advantage to using the RDO is speed! This is without a doubt one of the fastest methods for manipulating data. You also get the benefit of portability. You can run this methodology against any of the ODBC compliant databases.

The main disadvantages are the learning curve and having to write more code.

```
        Str2Null((txtLast)) & " "
    strSQL = strSQL & "WHERE iCust_id = " & txtCustID
End If

    ' Do the updating
    goRDOConn.Execute strSQL
    ' Check to see if rows were affected
    If goRDOConn.RowsAffected <> 1 Then
        MsgBox "Could not save information"
    End If
End If

    ' Refresh result set
    Call lstCustLoad

    ' Reset buttons
    Call ButtonReset
End Sub
Listing 2-36: FormSave() Routine.
```

The FormSave() routine is a little more difficult to implement. Different ODBC drivers support different functionality. Some implement Updateable cursors and some do not. In the FormSave() routine, the Updatable property of the ResultSet object determines if you can use the AddNew, Edit, and Update methods, just like in Jet. If you cannot use the methods, use straight SQL statements and submit the statement via the rdoConnection object using its Execute method.

```

Private Sub FormSave()
    Dim strSQL As String

    ' If ResultSet is updatable use objects
    If mrdoRes.Updatable Then
        If mboolAdding Then
            mrdoRes.AddNew
        Else
            mrdoRes.Edit
        End If
        ' Save all fields to dynaset
        mrdoRes!szCompany_nm = txtCompany
        mrdoRes!sFirst_nm = txtFirst
        mrdoRes!szLast_nm = txtLast
        mrdoRes!szStreet_ad = txtStreet
        mrdoRes!szCity_nm = txtCity
        mrdoRes!strState_cd = cboState
        mrdoRes!sZip_cd = txtZipCode

        ' Update dynaset
        mrdoRes.Update

        If mboolAdding Then
            mboolAdding = False
            mrdoRes.MoveLast
            txtCustID = mrdoRes!iCust_id
        End If
    Else
        If mboolAdding Then
            strSQL = "INSERT INTO tblCustomers ("
            strSQL = strSQL & "szCompany_nm, "
            strSQL = strSQL & "szStreet_ad, "
            strSQL = strSQL & "szCity_nm, "
            strSQL = strSQL & "strState_cd, "
            strSQL = strSQL & "sZip_cd, "
            strSQL = strSQL & "sFirst_nm, "
            strSQL = strSQL & "szLast_nm) "
            strSQL = strSQL & "Values("
            strSQL = strSQL & Str2Null((txtCompany)) & ", "
            strSQL = strSQL & Str2Null((txtStreet)) & ", "
            strSQL = strSQL & Str2Null((txtCity)) & ", "
            strSQL = strSQL & Str2Null((cboState)) & ", "
            strSQL = strSQL & Str2Null((txtZipCode)) & ", "
            strSQL = strSQL & Str2Null((txtFirst)) & ", "
            strSQL = strSQL & Str2Null((txtLast)) & ") "
        Else
            strSQL = "UPDATE tblCustomers SET "
            strSQL = strSQL & "szCompany_nm = " & _
                Str2Null((txtCompany)) & ", " & _
            strSQL = strSQL & "szStreet_ad = " & _
                Str2Null((txtStreet)) & ", " & _
            strSQL = strSQL & "szCity_nm = " & _
                Str2Null((txtCity)) & ", " & _
            strSQL = strSQL & "strState_cd = " & _
                Str2Null((cboState)) & ", " & _
            strSQL = strSQL & "sZip_cd = " & _
                Str2Null((txtZipCode)) & ", " & _
            strSQL = strSQL & "sFirst_nm = " & _
                Str2Null((txtFirst)) & ", " & _
            strSQL = strSQL & "szLast_nm = " & _

```

```
Private Sub FormShow()  
    mboolClick = False  
  
    If mrdoRes.EOF Then  
        mboolAddRecords = True  
    Else  
        txtCustID = mrdoRes!iCust_id  
        txtCompany = mrdoRes!szCompany_nm  
        txtFirst = mrdoRes!sFirst_nm  
        txtLast = mrdoRes!szLast_nm  
        txtStreet = mrdoRes!szStreet_ad  
        txtCity = mrdoRes!szCity_nm  
        txtZipCode = mrdoRes!sZip_cd  
  
        ' Find state code in combo box  
        Call ListFindString(cboState, mrdoRes!strState_cd  
  
        ' Reset buttons  
        Call ButtonReset  
    End If  
  
    mboolClick = True  
End Sub  
Listing 2-35: FormShow() Routine.
```

Saving Changes

If the user edits or adds customers, you need to save those changes to the table. When the user presses the Save button, call a routine called FormSave:


```

' If result set is open, close it, and reopen
If Not (mrdoRes Is Nothing) Then
    mrdoRes.Close
End If

 strSQL = "SELECT * FROM tblCustomers "
 strSQL = strSQL & "ORDER BY szCompany_nm "
Set mrdoRes = goRDOConn.OpenResultset(strSQL, rdOpenStatic)

' Load the data
Do Until mrdoRes.EOF
    lstCust.AddItem mrdoRes!szCompany_nm
    lstCust.ItemData(lstCust.NewIndex) = mrdoRes!iCust_id
    mrdoRes.MoveNext
Loop

' Fire click event on list box
If lstCust.ListCount > 0 Then
    lstCust.ListIndex = 0
End If
End Sub

```

Listing 2-33: Loading The Customer List Box.

Besides adding the customer name to the list box, you have loaded the customer table's primary key, `iCust_id`, into the `ItemData()` property. The key is useful when updating and deleting data from the customer table.

Displaying A Customer

When the user clicks on the List box, display the information on the form that corresponds to the company in the list box. Use the `AbsolutePosition` property of the `ResultSet` object to move to a particular record in the `ResultSet`. By setting the `AbsolutePosition` property of the `ResultSet` to a number, the current row pointer is updated to that row. Since you loaded the data into the list box, each element in the list box has a `ListIndex` position. Use this position to give you the `AbsolutePosition` of the row in the `ResultSet`:

```

Private Sub lstCust_Click()
    If lstCust.ListIndex <> -1 Then
        mrdoRes.AbsolutePosition = lstCust.ListIndex + 1
        Call FormShow
    End If
End Sub

```

Listing 2-34: Clicking On The Customer List Box.

After setting the `AbsolutePosition`, you need to call the `FormShow` routine to display the data in the current row in the text boxes on the form:

Loading The State Combo Box

After opening a connection to the data source, you can create a Resultset object and use this object to populate the form. Notice that the code looks very similar to the Jet DAO. Load up a combo box of state codes as follows:

```
Private Sub cboStateLoad()  
    Dim rdoRes As rdoResultset  
    Dim strSQL As String  
  
    strSQL = "SELECT strState_cd FROM tblStates"  
    Set rdoRes = goRDOConn.OpenResultset(strSQL)  
    Do Until rdoRes.EOF  
        cboState.AddItem rdoRes!strState_cd  
        rdoRes.MoveNext  
    Loop  
    rdoRes.Close  
    Set rdoRes = Nothing  
End Sub
```

Listing 2-32: cboStateLoad() Routine.

In the above routine, you dimmed a variable as a rdoResultSet object. You then set that equal to the return value from the OpenResultSet() method on the connection handle.

Loading The Customer List Box

In the General Declarations area of the Customer form, you dimmed a variable as an rdoResultSet object called mrdoRes. You can now use this object in multiple routines in the customer form.

First, set the object equal to the result of querying all customers in the tblCustomers table. Next, load up the list box, then use the object to display customers.

```
Private Sub lstCustLoad()  
    Dim strSQL As String  
  
    ' Clear any customers  
    lstCust.Clear
```

```

Private Sub Form_Load()
    ' Initialize form
    Call FormInit

    If RDOOpen() Then
        ' Load states
        Call cboStateLoad

        ' Load customers
        Call lstCustLoad
    End If
End Sub
Listing 2-29: Form_Load() Event.

```

In the General Declarations of the RDO.BAS file, you create a Public variable called goRDOConn to hold the connection handle:

```

Option Explicit
Public goRDOConn As rdoConnection
Listing 2-30: Public Connection Handle.

```

You can now use that rdoConnection object to open a connection to the data source:

```

Function RDOOpen() As Integer
    On Error GoTo RDOOpen_EH

    ' You can use the following
    ' rdoEngine.rdoDefaultCursorDriver = rdUseOdbc
    ' Set goRDOEnv = rdoEngine.rdoCreateEnvironment( _
    '     "Customer", "sa", "" )
    ' Set goRDOConn = goRDOEnv.OpenConnection( _
    '     "SQL60", rdDriverNoPrompt, _
    '     False, "UID=sa;PWD=" )

    ' or just do this
    Set goRDOConn = rdoEngine(0).OpenConnection( _
        "SQL60", rdDriverNoPrompt, _
        False, "UID=sa;PWD=" )

    RDOOpen = True

    Exit Function

RDOOpen_EH:
    MsgBox Err
    RDOOpen = False
    Exit Function
End Function
Listing 2-31: RDOOpen() Function.

```

The rdoEngine(0) object tells the RDO objects that you are using the first environment space. Alternatively, you can use the CreateEnvironment() method to create a new environment space. This is in the code that is commented out above. Use the OpenConnection() method to pass the name of the data source, "SQL60", to tell the system there are no prompts from the ODBC driver, and to pass the User ID and Password to the SQL Server.

Methods Of The Remote Data Objects

Object	Method	Description
rdoConnecti on	CreatePreparedStatem ent	Allows you to create a prepared statement that can be called over and over again with replaceable parameters. Think of this like a temporary stored procedure that accepts parameters.
	Execute	Executes an action query such as an INSERT, UPDATE or DELETE statement.
	OpenResultSet	Creates an rdoResultset object.
rdoResultSe t	GetRows	Returns a set of rows into a two dimensional array.
	MoreResults	Clears the current result set and returns a true if there are additional result sets to be processed.

Table 2-6: Important Methods For The Remote Data Objects.

Following is a look at a real world example of using some of the functionality of the RDO:

Figure 2-12: Customer Form Using The RDO Model.

This example will show how to build a typical customer information form that uses the RDO objects and methods and to add, edit, and delete records.

The first thing to do is open a connection to the data source, which in this case is SQL Server 6.x. This example assumes that you have exported the tables used in this seminar from the sample database to the SQL Server, and that you have created an ODBC data source to point to this database. In the Form_Load() event of this form, call the routine that opens the data source.

Remote Data Access Objects

Besides the Remote Data control, you also have a complete set of objects that deal just with remote data. These objects and collections are similar to the Jet DAO, but with a limited subset of the functionality.

Collection	Object	Description
None	rdoEngine	Similar to the DBEngine in Jet. It is the hook to the remote database engine.
rdoEnvironments	rdoEnvironment	The environment of the ODBC driver used for one or many connections to a data source.
rdoConnections	rdoConnection	A connection handle.
rdoResultsets	rdoResultset	Similar to a Dynaset or Snapshot in Jet. It is the rows of data returned from a query.

Table 2-4: Table Of Some RDO Objects.

The above table lists some of the most important objects and collections in the RDO model.

Properties Of The Remote Data Objects

Many of the properties for the remote data objects are the same as those you saw for the Remote Data control. Listed in the table below are those that you may set that are different from the Remote Data control or are different from the Jet DAO.

Object	Property	Description
rdoEnvironment	hEnv	The underlying ODBC Environment handle.
RdoConnection	AsyncCheckInterval	The number of milliseconds to check to see if an Asynchronous query is complete or not.
	hDbc	The underlying ODBC database connection handle.
	RowsAffected	The number of rows affected by the last INSERT, UPDATE or DELETE statement.
RdoResultset	hStmt	The underlying ODBC statement handle.
	Restartable	Returns a true if the resultset supports the Requery method.

Table 2-5: Important Properties Of The Remote Data Objects.

Canceling An Add/Edit

To cancel an Add or Edit operation you need to invoke either the CancelUpdate method on the Resultset object or the UpdateControls on the Remote Data control object:

```
Private Sub cmdClose_Click()  
    If cmdClose.Caption = "&Close" Then  
        Unload Me  
    Else  
        ' Cancel the Add/Edit  
        If mboolAdding Then  
            rdcCust.Resultset.CancelUpdate  
        Else  
            rdcCust.UpdateControls  
        End If  
        Call ButtonReset  
    End If  
End Sub  
Listing 2-27: cmdClose_Click() Event.
```

Deleting Data

To delete the currently highlighted row in the list box, the user may click the Delete button.

```
Private Sub cmdDelete_Click()  
    Dim intMsg As Integer  
  
    Beep  
    intMsg = MsgBox("Delete The Current Record", _  
        vbYesNo + vbQuestion, "Delete Record")  
    If intMsg = vbYes Then  
        ' Delete record  
        rdcCust.Resultset.Delete  
        rdcCust.Refresh  
    End If  
End Sub  
Listing 2-28: cmdDelete_Click() Event.
```

It is standard practice to ask the users if they really wish to delete the currently highlighted record. If they answer Yes, invoke the Delete method on the Recordset object. Once again, refresh the data using the Refresh method on the Remote Data control.

StillExecuting	Return a true if the query is still executing on the database. This is used with Asynchronous queries.
Transactions	Returns true if the data source supports transaction processing.
UserName	Set this property if you have not used the UID in the Connect property.
Version	Returns a value that specifies the version of the data source being used.

Table 2-3: Important Properties For the Remote Data control.

Adding Data

To add data using the Remote Data control, add the following code under the Click event of the New command button:

```
Private Sub cmdNew_Click()
    mboolAdding = True
    ' Perform an AddNew
    rdcCust.Resultset.AddNew
    ' Toggle buttons
    Call DataHasChanged
    txtCustID.SetFocus
End Sub
```

Listing 2-25: cmdNew_Click() Event.

To begin the adding process, first invoke the AddNew method on the Resultset property of the Data control. This is different from the Jet Data control that had a Recordset property. Toggle the buttons so only Save and Cancel are activated, and set focus to the Customer ID text box.

Saving The Data

To save the data when the user clicks on the Save button, invoke the UpdateRecord method on the Data control.

```
Private Sub cmdSave_Click()
    If mboolAdding Then
        rdcCust.Resultset.Update
    Else
        rdcCust.UpdateRow
    End If
    Call ButtonReset
End Sub
```

Listing 2-26: cmdSave_Click() Event.

Within the click event, you need to invoke different methods depending on whether you are adding or editing a row. Use the Update method on the Resultset object when adding, and use an UpdateRow method when editing.

Property	Description
Connection	A reference to the underlying rdoConnection object.
Connect	A valid connect string as determined by the ODBC Driver. For example, you may pass the DSN, UID, PWD or DATABASE to the Microsoft SQL Server Driver.
CursorDriver	May set to one of three values. You may specify for the ODBC driver to determine which cursor mode to use. You may specify to use the ODBC Cursor Library. Or you may specify to use Server-side cursors available with SQL Server 6.0 or Oracle 7.x.
DataSourceName	May be left blank if the DSN is filled in using the Connect property.
Environment	Returns a reference to the underlying rdoEnvironment object.
KeySetSize	If the default value of 0 is chosen you will get a keyset driven cursor only. This is the only type of cursor available with ODBC 2.5 or earlier.
LockType	You may set this to determine if you wish to have a read-only cursor, use pessimistic locking or use optimistic locking.
LoginTimeout	Set this to the number of seconds for ODBC to wait for a valid connection. You may need to set this higher on slower networks or busy servers.
MaxRows	May be set to the maximum amount of rows to return from a query.
Options	Used to specify Asynchronous or Synchronous query processing.
Password	May be used if the PWD option in the Connect property is left blank.
Prompt	Set to true to have the ODBC driver manager prompt for any missing connection information.
QueryTimeout	Specifies the number of seconds to wait before the ODBC driver manager times out a query and returns an error value to the calling program.
ReadOnly	Returns TRUE if the underlying cursor is read-only.
Resultset	Returns a reference to the underlying rdoResultset object. May also be used to set an existing rdoResultset to the data control.
ResultsetType	Returns or sets the type of cursor that is/will be opened. You may specify a 1 for a Keyset type. This is an editable cursor, but any changes made in the cursor show up as holes in another users cursor. You may specify 3 for a cursor that is similar to a Snapshot object, but is updatable. Changes made to this type of cursor are not reflected in other user's cursors.
RowsetSize	The number of rows of data that will be buffered on the client machine after a cursor is opened. The default is 100.
SQL	The SQL Statement that will be passed to the data source for processing. This may be a back-end specific SQL statement or one with ODBC escape clauses in it.

Properties Of The Remote Data control

Here are some of the properties of the Remote Data control.

7. Fill in the DataField property with the name of a field in the table. Click the down arrow on the DataField property to log in to the data source. The RDO then retrieves all of the columns from the table.
8. Run the project. The data from the ODBC data source displays.
9. Add, edit, and delete rows in the table as needed, just like a normal data control.

Once you have placed this control on the form, the coding is very similar to the method used for the Jet Data control.

Remote Data control

The Enterprise Edition of Visual Basic 4.0 has a new type of data control for use only with ODBC data sources. This data control is a very small wrapper around the ODBC API functions. This means that you are *not* going to use Jet to get at Microsoft SQL Server, Oracle 7 or other SQL databases. Using the Remote Data control and Remote Data Objects can improve retrieval performance 2-3 times over Jet.

The Remote Data control looks like the following in the Toolbox:



Figure 2-10: Remote Data control.

Figure 2-11: Remote Data control Sample.

To use the Remote Data control, follow the steps below.

1. Draw the Remote Data control on a form.
2. Press F4 and set the Name property to rdo<Something>. Give it a name that is meaningful.
3. Set the DataSourceName property to a DSN that you created in the ODBC Administrator.
4. Fill in the SQL property with a SELECT statement that retrieves information from a table in the ODBC data source.
5. Add the text boxes and labels appropriate for the database/table from which data is being retrieved.
6. On each text box, fill in the DataSource property with the name of the RDO control.

```
Sub ODBCUnload(hdbc As Long, hEnv As Long)
    Dim intStatus As Integer

    If hdbc <> 0 Then
        intStatus = SQLDisconnect(hdbc)
        If intStatus = SQL_ERROR Then
            MsgBox "Error logging out of data source!"
        End If
    End If

    intStatus = SQLFreeConnect(hdbc)
    If intStatus = SQL_ERROR Then
        MsgBox "Error freeing connection handle"
    End If

    If hEnv <> 0 Then
        intStatus = SQLFreeEnv(hEnv)
        If intStatus = SQL_ERROR Then
            MsgBox "Error freeing environment from ODBC drivers"
        End If
    End If
End Sub
```

Listing 2-24: ODBCUnload() Routine.

The routine above calls three ODBC API functions: SQLDisconnect() to drop the connection to the database, SQLFreeConnect() to free the memory associated with the connection handle, and SQLFreeEnv() to free the memory for the environment and unload the Driver Manager dynamic link library from memory.

```
' Add to combo box
cboState.AddItem strState

' Get next row
intStatus = SQLFetch(hStmt)
Loop
End If

' Free the statement handle
intStatus = SQLFreeStmt(hStmt, SQL_DROP)
End Sub
```

Listing 2-23: cboStateLoad() Procedure.

In the above code, you call ODBCQuerySubmit to create a Statement handle and send the SQL statement to the server for processing. This routine calls SQLAllocStmt() and SQLExecDirect(). Use the SQLFetch() function to position the cursor to the first row in the result set. Call ODBCData(), a wrapper function, to retrieve one column of data from the result set. ODBCData() calls the SQLGetData() function.

After you retrieve all of the rows from the result set, you need to close the statement handle and release all of the memory associated with that handle. You can do this by calling the SQLFreeStmt() function.

Adding/Editing/Deleting Data

Most of the other routines for saving and deleting data are very similar to the code written in the Jet DAO section. You can use the ODBCQuerySubmit() function to submit the appropriate SQL statements. You need to perform all of the INSERT, UPDATE, and DELETE statements when using the ODBC API. There are no object wrappers to make the job easier.

Closing The Connection

When you are finished with the ODBC data source, you must make sure to close the connection and free all the memory allocated in the beginning. In the Form_Unload() event, you can call a routine called ODBCUnload():

Initializing And Connecting To ODBC

In the sample application you'll noticed that there are many wrappers around the functions. This helps hide many of the details of dealing with the ODBC API and helps keep the code more readable. Now look at the application, and some of the wrapper functions.

The first thing you need to do is to open a connection to the database:

```
Private Sub Form_Load()  
    ' Initialize form  
    Call FormInit  
  
    If ODBCInit(ghEnv, ghdbc) Then  
        If ODBCDataOpen(ghdbc) Then  
            ' Load states  
            Call cboStateLoad  
  
            ' Load customers  
            Call lstCustLoad  
        End If  
    End If  
End Sub  
Listing 2-22: Form_Load() Event.
```

The two wrapper functions ODBCInit() and ODBCDataOpen() perform the necessary allocation of memory and connecting to the data source. These two functions perform the following calls:

- SQLAllocEnv()
- SQLAllocConnect()
- SQLConnect()

Once you have a connection, you can load the combo box of state codes and all of the customers:

```
Private Sub cboStateLoad()  
    Const STATE_CODE As Integer = 1  
  
    Dim intStatus As Integer  
    Dim strSQL As String  
    Dim strState As String  
    Dim hStmt As Long  
  
    ' Load list box  
    strSQL = "SELECT strState_cd FROM tblStates"  
    If ODBCQuerySubmit(ghdbc, hStmt, strSQL) Then  
        ' Get first Row  
        intStatus = SQLFetch(hStmt)  
        Do While intStatus = SQL_SUCCESS  
            ' Get state code  
            strState = ODBCData(hStmt, STATE_CODE)
```

How Visual Basic And ODBC Communicate

Before using the ODBC API, you need to understand how to communicate with the ODBC drivers. First you load the ODBC driver manager, which in turn loads the particular driver you are interested in using. When you establish a connection to the data source, the connection is returned to your Visual Basic program as a **Long** integer. This long integer is called a *connection handle* to the data source. Almost all of the ODBC API functions use this handle to submit queries and retrieve data from the data source.

Most ODBC API calls return a status indicating what happened in the function call. The status is either an error code or a success code. Any data you need from the function is passed by reference in an argument. For example:

```
Sub ODBCInit() as integer
    Dim intRet As Integer
    Dim hEnv As Long
    intRet = SQLAllocEnv(hEnv)
    If intRet <> 0 Then
        MsgBox "Unable to initialize ODBC API drivers!"
        ODBCInit = False
    End If
End Sub
Listing 2-21: ODBCInit().
```

The variable *hEnv* is initialized to 0 in the **Dim** Statement, then passed to the SQLAllocEnv() function by reference. If everything is successful, *hEnv* is a different value after the function has executed. The SQLAllocEnv() function returns an Integer value indicating the success or failure of the function call.

Use the following code to create two global variables that hold the environment handle and the database connection handle:

```
Global ghEnv As Long
Global ghdbc As Long
```

NOTE: When you see ODBC as a prefix to a function name during this class, you can assume it is one of the wrapper functions. All the functions with a prefix of SQL are actual ODBC API calls.

ODBC Functions

The following table is a list of the most frequently used functions in the ODBC API.

Function Name	Description
SQLAllocEnv()	Sets aside memory for ODBC Environment handle.
SQLAllocConnect()	Sets aside memory for a connection handle.
SQLDriverConnect()	Connect to a data source such as SQL Server or Microsoft Access.
SQLAllocStmt()	Sets aside memory for a SQL statement handle.
SQLExecDirect()	Sends the SQL statement to the driver.
SQLFetch()	Retrieves one row of data from the statement handle. Only retrieves one row at a time.
SQLGetData()	Retrieves one column of data from the current row in the statement handle.
SQLFreeStmt()	Releases the memory used by the statement handle.
SQLDisconnect()	Logs out of the data source.
SQLFreeConnect()	Releases the memory used by the connection.
SQLFreeEnv()	Releases the memory for the ODBC environment.

Table 2-2: Frequently Used ODBC API Calls.

Here is an example of using the ODBC API.

Customer Information using ODBC API

Paul's Boat Shop
 Paul's Picture Company
 Pick-Me-Up Coffee Supplies
 Ricky's Bakery
 Ronald Software
 Sally's Salmon Hatchery
 Sam's Silver Mine
 Sue's Bike Shop
 Tim's Tractors
 Tom's Consulting
 Walter's Windows
Widget Factory
 XYZ Corporation

Customer ID: 14
 Company: Widget Factory
 First Name: Donald
 Last Name: Williams
 Street: 100 Queens Blvd., Suite 50
 City: Madison
 State: WI Zip Code: 42959

[New] [Delete] [Save] [Close]

Figure 2-9: Customer Form Using The ODBC API.

Using The ODBC API

Another database access method is the ODBC Application Programming Interface (API). This method bypasses the Jet engine. There are several functions that you can use to open a database connection and retrieve data from a data source.

ODBC Function Overview

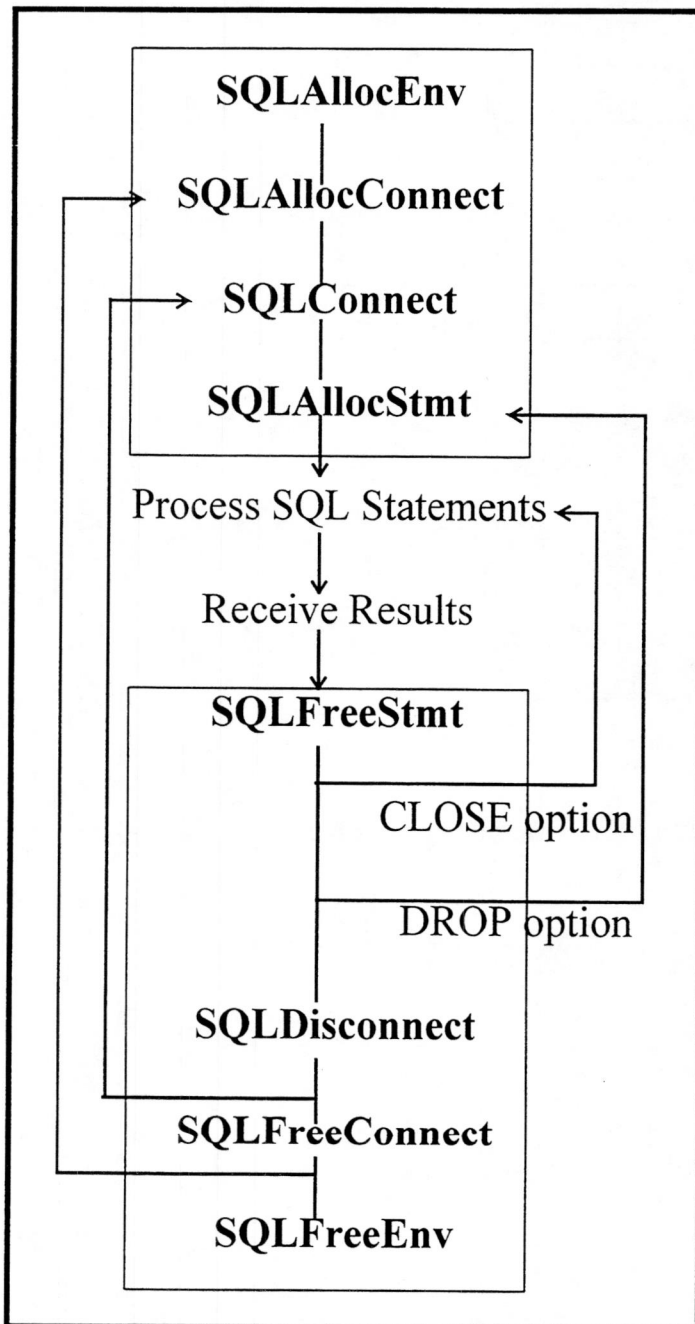


Figure 2-9: ODBC Function Overview

Create a unique Data Source Name and select a Server. If there is more than one SQL Server on the network, the names display in a drop down combo box. You can also set up options for the SQL Server by clicking on the Options button.

The screenshot shows the 'ODBC SQL Server Setup' dialog box. It has a title bar with a minus button and the text 'ODBC SQL Server Setup'. The dialog is divided into several sections. At the top, there are three buttons: 'OK', 'Cancel', and 'Help'. Below these are input fields for 'Data Source Name:', 'Description:', 'Server:', 'Network Address:', and 'Network Library:'. The 'Server:' field has a dropdown arrow. The 'Network Address:' and 'Network Library:' fields show '(Default)'. To the right of these fields is an 'Options >>' button. Below these fields is a 'Login' section with a 'Database Name:' field, a 'Language Name:' dropdown (showing '(Default)'), and a checked checkbox for 'Generate Stored Procedure for Prepared Statement'. At the bottom is a 'Translation' section with an unchecked checkbox for 'Convert OEM to ANSI characters' and a 'Select...' button.

Figure 2-8: Options For SQL Server.

In the bottom half of this dialog box under Login, you can specify the default Database Name to use when logging on.

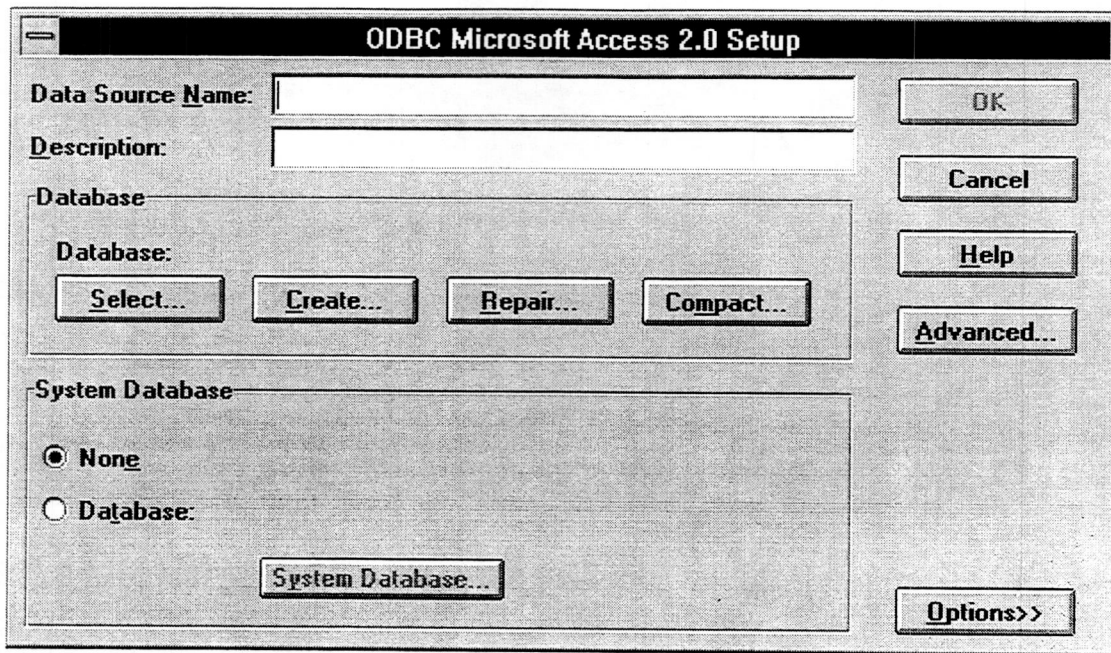


Figure 2-6: ODBC Microsoft Access 2.0 Setup Dialog Box.

Fill in the Data Source Name with a unique name, then Select the database. If you have a System database that needs to be used with this database, click the Database option, then click System Database.

Adding A New Microsoft SQL Server Data Source

To create a new Microsoft SQL Server data source, click the Add button, and select the SQL Server driver. The following dialog box displays.

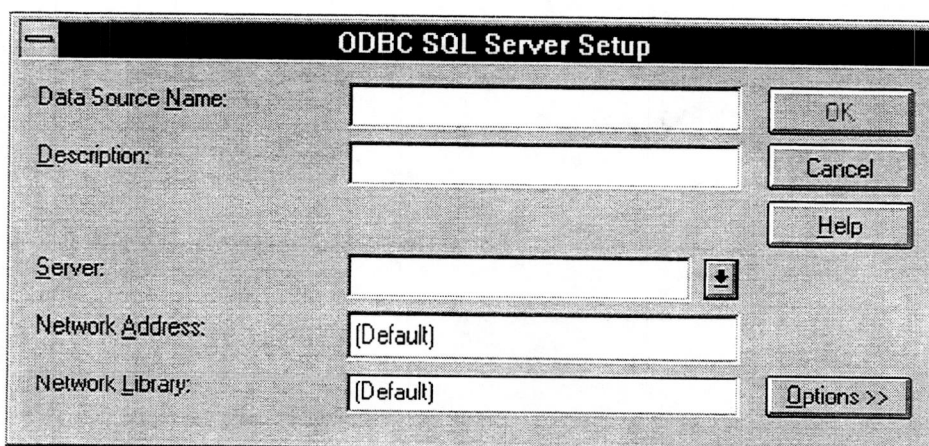


Figure 2-7: ODBC SQL Server Setup Dialog Box.

This box differs from the Microsoft Access driver dialog since Microsoft SQL Server needs different information from what Microsoft Access needs.

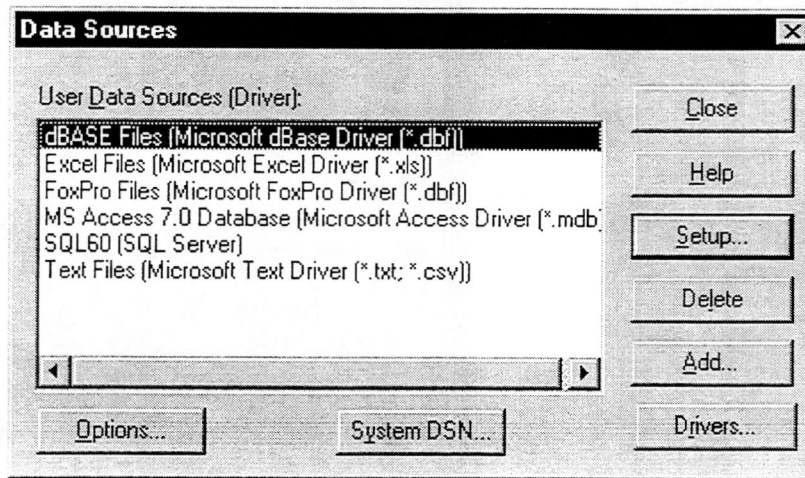


Figure 2-4: ODBC Data Sources Dialog Box.

Once the data source name is contained in this box, you can access it from your Visual Basic application.

Adding A New Microsoft Access Data Source

From the ODBC Data Sources dialog box, click the Add button to add a new data source name. A dialog box that looks like the following displays:

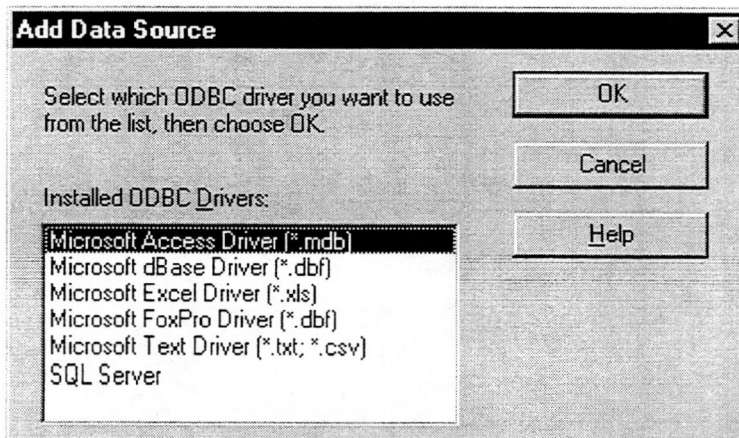


Figure 2-5: Add Data Source Dialog Box.

This is where you choose the ODBC driver you wish to use. Select Microsoft Access Driver (*.mdb) from the list box and click the OK button. The ODBC Microsoft Access 2.0 Setup dialog box displays.

ODBC Setup

To install ODBC on your system, you need to have a Microsoft product that supports ODBC. These products include Microsoft Access 1.1, Microsoft Access 2.0, Microsoft Access 95, Excel 5.0, Excel 7.0, Visual Basic 3.0, and several others. When you use one of the products to install ODBC, the following files are added to your WINDOWS directory:

- ODBC.INI is the ODBC driver information for driver manager.
- ODBCINST.INI provides the list of ODBC drivers installed on your machine.

WARNING! NEVER modify ODBC.INI yourself. Always use the ODBC Administrator Utility to change this file.

The ODBC installation program places the following files in your WINDOWS\SYSTEM directory:

- ODBC.DLL is the driver manager for ODBC.
- ODBCADM.EXE is the ODBC administrator, used to add, modify and delete ODBC drivers.
- ODBCINST.DLL is the file with the ODBC installation procedures.
- ODBCINST.HLP is the Help file for the ODBC installation program.

After installing the ODBC drivers, the ODBC icon is added to the Control Panel. Select the Control Panel from the Main group in Program Manager. Double-click the ODBC icon and use the Data Sources dialog box to add, modify or delete information about the data sources you have installed on your system.

ODBC In Visual Basic

When you are using Visual Basic, the Jet Engine actually sits between Visual Basic and the ODBC interface. The Jet engine performs some translations before giving the SQL statement to ODBC for its translation and is responsible for calling the ODBC API functions. Visual Basic may also call the ODBC API directly. This is covered later.

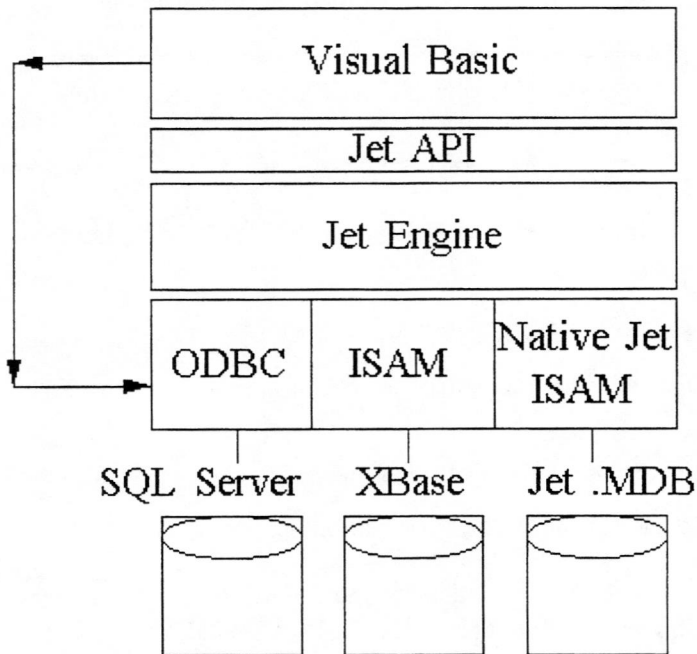


Figure 2-3: ODBC Configuration In Visual Basic.

ODBC Components

There are four components in the ODBC architecture, as shown below.

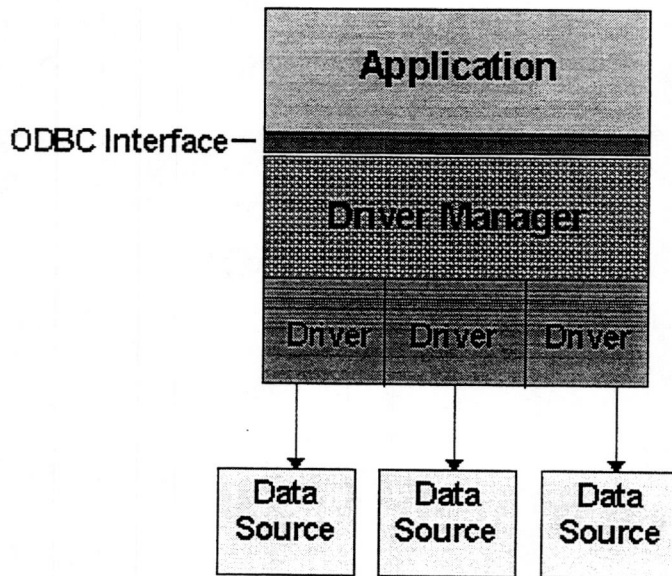


Figure 2-2: ODBC Configuration.

1. **Application:** Your application calls ODBC functions to transmit SQL statements to the back end for processing, and to retrieve a result set.
2. **Driver Manager:** The driver manager loads the appropriate ODBC driver requested by the application.
3. **Driver:** The ODBC driver processes ODBC function calls, performs the translations required to guarantee that the back end recognizes the SQL, and returns the result set to the application.
4. **Data Source:** The data source is the database manager, the data itself, and the operating system and network required to access the database manager.

It is a good idea to use Microsoft Access MDB files while your application is in the prototyping and development phase. This makes it easy to give demos to customers on a laptop computer. All data access routines should be written in ODBC-SQL. When you are ready to deliver the application, you may change the ODBC driver and use SQL Server for the final production application. The ODBC driver for SQL Server translates the SQL statements into the appropriate syntax for SQL Server.

<p>TIP: If you use the Jet Engine for your development, use Microsoft Access SQL syntax instead of ODBC-SQL syntax.</p>
--

ODBC

Open Database Connectivity (ODBC) is a standard proposed by Microsoft that allows a single program to connect to multiple back ends through a common programming API. This API allows you to pass one set of SQL statements to the ODBC interface to be translated into the appropriate dialect for the back end being used.

Purpose Of ODBC

The ODBC interface converts a generic form of SQL into the SQL that is supported by a particular vendor's DBMS. The ODBC API is an interface that runs on the client side of an application. The major advantage of using the ODBC interface is that you can write one set of source code, change the ODBC driver, and have your application talk to a different back end.

For example, if you were using Microsoft Access SQL to perform an outer join between a customers table and a contacts table, you would write:

Microsoft Access SQL:

```
SELECT tblCustomers.sCompany_nm, tblInvoice.szInvoice_desc  
FROM tblCustomers LEFT OUTER JOIN tblInvoice  
ON tblCustomers.lCust_id = tblInvoice.lCust_ID
```

Listing 2-18: Microsoft Access SELECT Statement.

Using Microsoft SQL Server syntax, however, you would write the code as follows:

Microsoft SQL Server Transact-SQL:

```
SELECT tblCustomers.sCompany_nm, tblInvoice.szInvoice_desc  
FROM tblCustomers, tblInvoice  
WHERE tblCustomers.lCust_id *= tblInvoice.lCust_id
```

Listing 2-19: Microsoft SQL Server SELECT Statement.

Finally using the ODBC API SQL, you would write the following:

ODBC SQL:

```
SELECT tblCustomers.sCompany_nm, tblInvoice.szInvoice_desc  
FROM { oj tblCustomers LEFT OUTER JOIN tblInvoice  
ON tblCustomers.lCust_id = tblInvoice.lCust_id }
```

Listing 2-20: ODBC SELECT Statement.

If you write your application using ODBC-SQL syntax, you can be assured your SQL statements will be translated correctly to the back end you decide to utilize.

Canceling An Add/Edit

To cancel either the Add or the Edit of the record on the form, redisplay the current row in the Recordset object. Since you have not performed an AddNew or Edit on the underlying Recordset object, you do not need to cancel anything.

```
Private Sub cmdClose_Click()  
    If cmdClose.Caption = "&Close" Then  
        Unload Me  
    Else  
        Call FormShow  
        Call ButtonReset  
        If mdsData.EOF Then  
            Call Form_Activate  
        End If  
    End If  
End Sub
```

Listing 2-16: cmdClose_Click() Event.

Deleting Data

Deleting the currently highlighted row is not much different from the Data control approach.

```
Private Sub cmdDelete_Click()  
    Dim intMsg As Integer  
  
    Beep  
    intMsg = MsgBox("Delete The Current Record", _  
        vbQuestion + vbYesNo, "Delete Record")  
    If intMsg = vbYes Then  
        mdsData.Delete  
        Call lstCustLoad  
    End If  
End Sub
```

Listing 2-17: cmdDelete_Click() Event.

It is standard practice to ask users if they really wish to delete the currently highlighted record. If they answer Yes, invoke the Delete method on the Recordset object.

```
' Save all fields to dynaset
mdsData!lCust_id = lngNext
mdsData!szCompany_nm = txtCompany
mdsData!sFirst_nm = txtFirst
mdsData!szLast_nm = txtLast
mdsData!szStreet_ad = txtStreet
mdsData!szCity_nm = txtCity
mdsData!strState_cd = cboState
mdsData!sZip_cd = txtZipCode

' Update dynaset
mdsData.Update

If mboolAdding Then
    mboolAdding = False
    mdsData.MoveLast
    txtCustID = mdsData!lCust_id
End If

' Refresh result set
Call lstCustLoad

' Reset buttons
Call ButtonReset
End Sub
Listing 2-15: FormSave() Procedure.
```

So that the application can generate a unique customer ID when adding a new record, the code calls a routine named TableIDGet(). A table named tblTableIDs is in our database that maintains the last customer ID assigned. When we call the TableIDGet(), it retrieves the next ID from this table, then increments it by one. In Microsoft Access you can use the Counter field to perform this automatic incrementing for you. However, if the application will be going against many different databases, you cannot guarantee that other databases will have a Counter type field.

Adding Data

To add a new row to the Recordset, you need to clear the text boxes on the form to give the user a blank screen for entry. This is accomplished by clicking on the New button:

```
Private Sub cmdNew_Click()  
    mboolAdding = True  
    Call FormClear  
    Call DataHasChanged  
    txtCompany.SetFocus  
End Sub  
Listing 2-12: cmdNew_Click() Event.
```

The routine, FormClear, is responsible for clearing all of the controls on this form:

```
Private Sub FormClear()  
    txtCustID = ""  
    txtCompany = ""  
    txtFirst = ""  
    txtLast = ""  
    txtStreet = ""  
    txtCity = ""  
    cboState.ListIndex = -1  
    txtZipCode = ""  
End Sub  
Listing 2-13: FormClear() Event.
```

Saving Data

When the user presses the Save button, you need to take all of the information from the controls on this form, and either add a new row, or edit the current row in the Recordset object.

```
Private Sub cmdSave_Click()  
    Call FormSave  
End Sub  
Listing 2-14: cmdSave_Click() Event.
```

The routine, FormSave, is actually responsible for performing the update of the Recordset:

```
Private Sub FormSave()  
    Dim lngNext As Long  
  
    If mboolAdding Then  
        ' If adding, get next ID for table  
        lngNext = TableIDGet("tblCustomers")  
    End If  
  
    If mboolAdding Then  
        mdsData.AddNew  
    Else  
        mdsData.Edit  
    End If
```

In the Click event of this list box, you will use the ListIndex property to give the absolute position of the row in the Recordset to move to. The AbsolutePosition is a new property of the Recordset object that gives you an ordinal position of the records within this object. You can also use the primary key, lCust_id, that was loaded into the ItemData() property of the list box.

After you are positioned on the row to be displayed, you need to display all the information from the Recordset. To accomplish this, call the FormShow routine:

```
Private Sub FormShow()  
    mboolClick = False  
  
    If mdsData.EOF Then  
        mboolAddRecords = True  
    Else  
        txtCustID = mdsData!lCust_id  
        txtCompany = Field2Str(mdsData!szCompany_nm)  
        txtFirst = Field2Str(mdsData!szFirst_nm)  
        txtLast = Field2Str(mdsData!szLast_nm)  
        txtStreet = Field2Str(mdsData!szStreet_ad)  
        txtCity = Field2Str(mdsData!szCity_nm)  
        txtZipCode = Field2Str(mdsData!sZip_cd)  
  
        ' Find state code in combo box  
        Call ListFindString(cboState, _  
            Field2Str(mdsData!strState_cd))  
  
        ' Reset buttons  
        Call ButtonReset  
    End If  
  
    mboolClick = True  
End Sub  
Listing 2-11: FormShow() Procedure.
```

The FormShow() procedure loads the information from the current row of the Recordset into each of the appropriate text boxes.

```
Private Sub cboStateLoad()  
    Dim snpData As Recordset  
    Dim strSQL As String  
  
    strSQL = "SELECT sState_cd FROM tblStates"  
    Set snpData = gdb.OpenRecordset(strSQL, _  
        dbOpenSnapshot)  
    Do Until snpData.EOF  
        cboState.AddItem snpData!strState_cd  
        snpData.MoveNext  
    Loop  
    snpData.Close  
    Set snpData = Nothing  
End Sub
```

Listing 2-8: cboStateLoad() Procedure.

Next, load all the customers into the list box.

```
Private Sub lstCustLoad()  
    Dim strSQL As String  
    ' Clear any customers  
    lstCust.Clear  
  
    ' If result set is open, close it, and reopen  
    If Not (mdsData Is Nothing) Then  
        mdsData.Close  
    End If  
  
    strSQL = "SELECT * FROM tblCustomers "  
    strSQL = strSQL & "ORDER BY szCompany_nm "  
    Set mdsData = gdb.OpenRecordset(strSQL, dbOpenDynaset)  
  
    ' Load the data  
    Do Until mdsData.EOF  
        lstCust.AddItem mdsData!szCompany_nm  
        lstCust.ItemData(lstCust.NewIndex) = _  
            mdsData!lCust_id  
        mdsData.MoveNext  
    Loop  
  
    ' Fire click event on list box  
    If lstCust.ListCount > 0 Then  
        lstCust.ListIndex = 0  
    End If  
End Sub
```

Listing 2-9: lstCustLoad() Procedure.

The last thing the routine does is set the ListIndex property of the list box. By doing this, you fire the Click event of that list box. Next, look at the Click event of the List Box control:

```
Private Sub lstCust_Click()  
    If lstCust.ListIndex <> -1 Then  
        mdsData.AbsolutePosition = lstCust.ListIndex  
        Call FormShow  
    End If  
End Sub
```

Listing 2-10: Click Event Of The Customer List Box.

Using The Data Access Objects

As an alternative to the Data control, you can program the Data Access Objects (DAO) yourself using either Visual Basic 4.0 Professional Edition or Visual Basic 4.0 Enterprise Edition. Now, look at the same customer form using DAO.

In the Form_Load() event of the customer form, you call a routine named JetOpen() to open the database. If the call is successful, you then load a combo box with state codes, and populate the list box of customers.

```
Private Sub Form_Load()  
    ' Initialize form  
    Call FormInit  
  
    If JetOpen() Then  
        ' Load states  
        Call cboStateLoad  
  
        ' Load customers  
        Call lstCustLoad  
    End If  
End Sub
```

Listing 2-6: Form_Load() Event.

The JetOpen() function checks to see if you are going against a server or not. If you are using Microsoft SQL Server, you need to build a connect string, otherwise you open a Microsoft Access MDB file.

```
Function JetOpen() As Integer  
    Dim strConnect As String  
  
    On Error GoTo JetOpen_EH  
  
    If gboolServer Then  
        strConnect = "ODBC;DSN=SQL60;UID=sa;PWD=;"  
        Set gdb = DBEngine(0).OpenDatabase("", False, _  
            False, strConnect)  
    Else  
        Set gdb = DBEngine(0).OpenDatabase(DATABASE_NAME)  
    End If  
  
    JetOpen = True  
  
    Exit Function  
  
JetOpen_EH:  
    MsgBox Err.Description  
    JetOpen = False  
    Exit Function  
End Function
```

Listing 2-7: JetOpen() Function.

After the database is open, load the state codes into a combo box using the following code:

```
    If mboolAdding Then
        ' Cancel the AddNew
        datCust.Recordset.CancelUpdate
    Else
        ' Refresh the controls
        datCust.UpdateControls
    End If
    ' Set the bound text
    Call ButtonReset
End If
End Sub
Listing 2-4: cmdClose_Click() Event.
```

Deleting Data

To delete the currently highlighted row in the list box, the user may click the Delete button.

```
Private Sub cmdDelete_Click()
    Dim intMsg As Integer

    Beep
    intMsg = MsgBox("Delete The Current Record",
        vbYesNo + vbQuestion, "Delete Record")
    If intMsg = vbYes Then
        ' Delete record
        datCust.Recordset.Delete
        datCust.Recordset.Requery
    End If
End Sub
Listing 2-5: cmdDelete_Click() Event.
```

It is standard practice to ask users if they really wish to delete the currently highlighted record. When they answer Yes, invoke the Delete method on the Recordset object. To make sure you have all the updated information in a multi-user environment you can invoke the Requery method on the Recordset object. This will requery all of the data that is physically on the table. Unfortunately this will also create network traffic, so be very careful when using this method.

Listing 2-1: Synchronize List Box And Data control.

By taking the SelectedItem and assigning it to the Bookmark property of the Data control, you force the Data control to update the Recordset object.

Adding Data

To add data using a Data control, add the following code under the Click event of the New command button:

```
Private Sub cmdNew_Click()  
    ' Perform an AddNew  
    datCust.Recordset.AddNew  
    ' Toggle buttons  
    Call DataHasChanged  
    txtCustID.SetFocus  
End Sub
```

Listing 2-2: cmdNew_Click() Event.

To begin the adding process, first invoke the AddNew method on the Recordset property of the Data control. Next, toggle the buttons so only Save and Cancel are activated. Finally, set focus to the Customer ID text box.

Saving The Data

To save the data when the user clicks on the Save button, simply invoke the UpdateRecord method on the Data control.

```
Private Sub cmdSave_Click()  
    datCust.UpdateRecord  
    Call ButtonReset  
End Sub
```

Listing 2-3: cmdSave_Click() Event.

The UpdateRecord method moves all the data from the data bound text boxes and updates the Recordset. It is a useful way to refresh the whole Recordset.

Canceling An Add/Edit

To cancel an AddNew you will invoke the CancelUpdate method on the Recordset object. This is a new method that was added with Visual Basic 4.0. To cancel an Edit, simply invoke the UpdateControls on the Data control. This will move the information in the Recordset into the controls on the form.

```
Private Sub cmdClose_Click()  
    If cmdClose.Caption = "&Close" Then  
        Unload Me  
    Else
```

Using The Data control

The Data control was first introduced with Visual Basic 3.0. There are many enhancements in the Visual Basic 4.0 Data control. It is an excellent tool for prototyping your application.

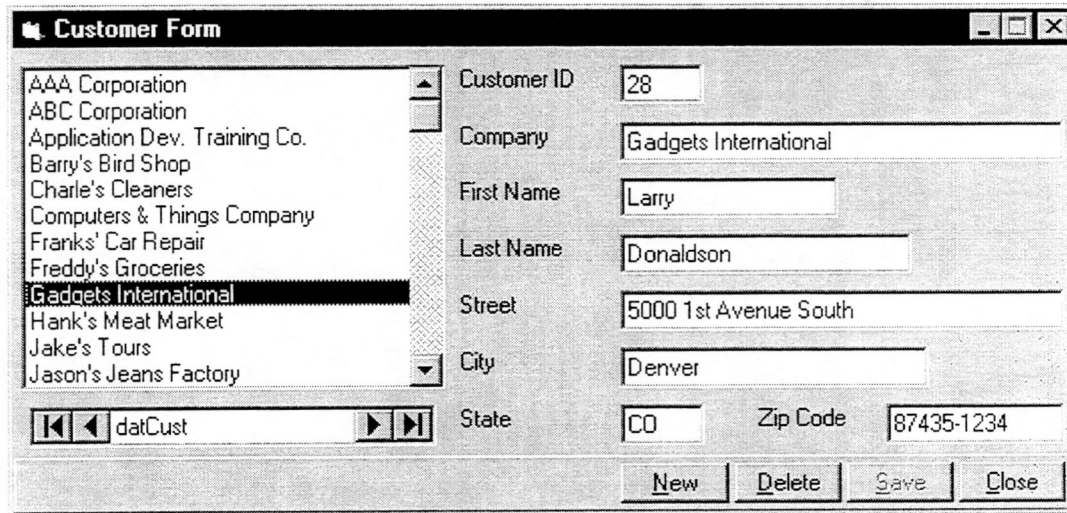


Figure 2-1: Customer Form Using A Data control.

Master Steps:

1. Place a Data control on a form.
2. Set the Name property to "datCust".
3. Set the DatabaseName property to "...\\...\\database\\Invoice.mdb".
4. Set the RecordSource property to "tblCustomers".
5. Place a Data Aware List Box on the form.
6. Set the Name property to "dblCust".
7. Set the RowSource property to "datCust".
8. Set the ListField property to "szCompany_nm".
9. Run the program.

The list box displays the company names as in the figure above.

Next, add the capability to click on an item in the list box and have the data aware controls update with the appropriate company information. There is a property on the Data Aware List Box called SelectedItem. This property contains the Bookmark of the row that corresponds to the item in the list box. In the Click event of the list box, add the following code:

```
Private Sub dblCust_Click()  
    datCust.Recordset.Bookmark = dblCust.SelectedItem  
End Sub
```

Database Programmability

There are many methods for accessing databases besides the Jet Engine. With the different versions of Visual Basic 4.0, the following data access methods are possible:

Method	Edition Available	Database Engine
Data Control	All	Jet Engine
Data Access Objects	Professional, Enterprise	Jet Engine
Remote Data Control	Enterprise (32-bit only)	ODBC
Remote Data Access Objects	Enterprise (32-bit only)	ODBC
ODBC API	All	ODBC
VB/SQL	All (SQL Server only)	Jet Engine

Table 2-1: Data Access Methods.

Introduction

This section explores many of the different options for database development using Visual Basic 4.0. You will explore a sample Customer application that was written using several different methodologies, and you will learn the advantages of each methodology.

Objectives

In this section you will:

- Learn about the different data access methods available with Visual Basic 4.0.
- See how to use the Data control.
- Learn how to use Data Access Objects.
- Discover what ODBC is.
- Learn to use the ODBC Application Programming Interface (API).
- See how to improve performance using the Remote Data control.
- Learn how to work with Remote Data Objects.
- Discuss the advantages of ODBC and Remote Data Objects (RDO).

Database Development

Checking Files In And Out

To check out a file, select Tools, Check Out from the Visual Basic menu. You will now see a check mark next to the files that you have checked out. Any files that are checked out by other people will appear with a gray box next to the file name. After you have made any changes to a source file, check the file back in using the Tools, Check In menu.

Other Options

Under the Add-ins menu, Visual SourceSafe will appear as a menu item. From this menu item you may perform the following tasks:

- Show History
- Show Differences
- SourceSafe Properties
- Add Files To SourceSafe
- Share Files
- Open New SourceSafe Project
- Run SourceSafe
- Options
- Refresh File Status

Each of these menus allows you to perform many of the most common tasks you would normally do in Visual SourceSafe without having to go into Visual SourceSafe.

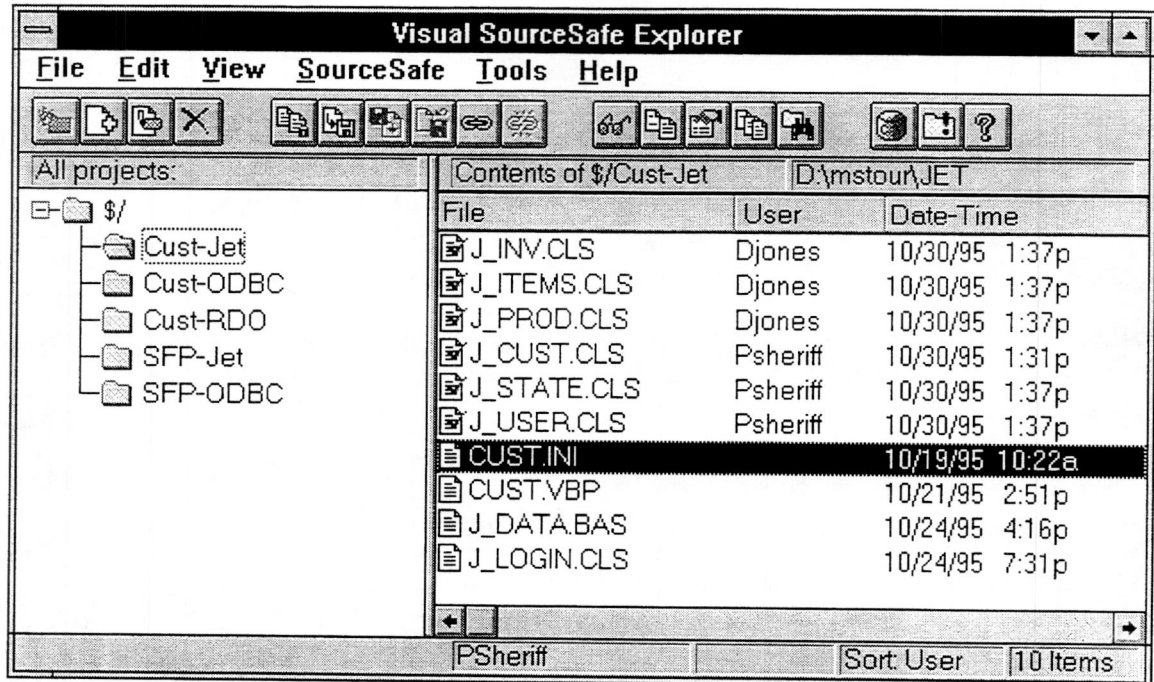


Figure 1-28: Visual SourceSafe Explorer.

Adding Files

After creating a project you will need to add the files from your project directory into the Project you created in Visual SourceSafe. Once added, each of the developers will be allowed to check these files in and out using the **T**ools menu from within Visual Basic. After the files have been added, you will see a box in your project file next to each file.

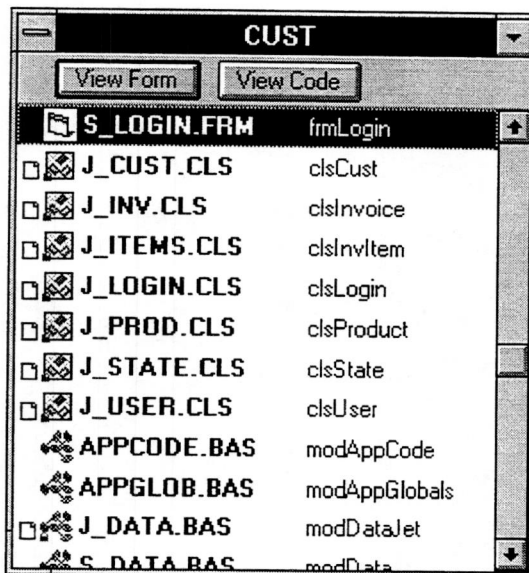


Figure 1-29: CUST Project Directory.

Version Control

The Enterprise Edition of Visual Basic includes a copy of Microsoft Visual SourceSafe, a version control system that allows you to check your projects, forms, .BAS and .CLS files into and out of a central repository. While this tool has traditionally been for multi-programmer shops, it can benefit the independent developer.

Visual SourceSafe integrates right into the Visual Basic development environment. You never have to leave your project to check files in and out. Here is how to setup a typical installation for a project.

Installation

First you need to install the Visual SourceSafe files from the Enterprise Edition CD-ROM. These files are located under the \SRCSAFE directory. Simply run the SETUP.EXE in this directory. You will be prompted to either perform a Server installation or a Client installation. If you will be placing all your files onto one server you should run the Server installation. After this is setup, everyone on the network can use this version to check files into and out of the system. However, to integrate Visual SourceSafe into Visual Basic, you will need to install the Client software on every developer's desktop. Once this is done, Visual SourceSafe will become an option under the Add-ins menu, and you will be able to use Get, Check In, Check Out, Undo Check Out from the Tools menu.

Adding Users

After installing Visual SourceSafe you need to add users to the system. This is done with the Visual SourceSafe 4.0 Admin icon. This icon is in the Microsoft Visual SourceSafe Folder. Add all your developer's names to this tool, and assign each one a password.

Creating A Project

Next you need to create a project name. This is a very important step because if you do not create a project name, Visual SourceSafe will put all your files into the main project. By creating projects, each project will show up under the Visual SourceSafe Explorer. Below is a sample of different projects, and the files checked out by different people. A project is created by selecting File, Create Project... from the main menu.

How It Works

When the user selects your add-in from the Add-in Manager menu, the `ConnectAddIn()` event is called in your OLE server, and is passed the instance handle of the Visual Basic environment. Use this instance handle to add a menu under the Add-ins menu. Next, tell the Visual Basic environment which object to call with an `AfterClick()` event when the user clicks on the menu. In the listing above, you used **Me** to have Visual Basic call the object. It is possible to create two or three menus to associate with two or three different objects. This can be done by using the class names instead of the `Me` keyword.

Once the Add-in has been registered, all the user has to do is click on the menu item. Your OLE server is called and its `AfterClick()` event fires. This is where you write the code to perform whatever action you wish to do.

Below is an overview of some of the objects that you can manipulate with the Add-ins.

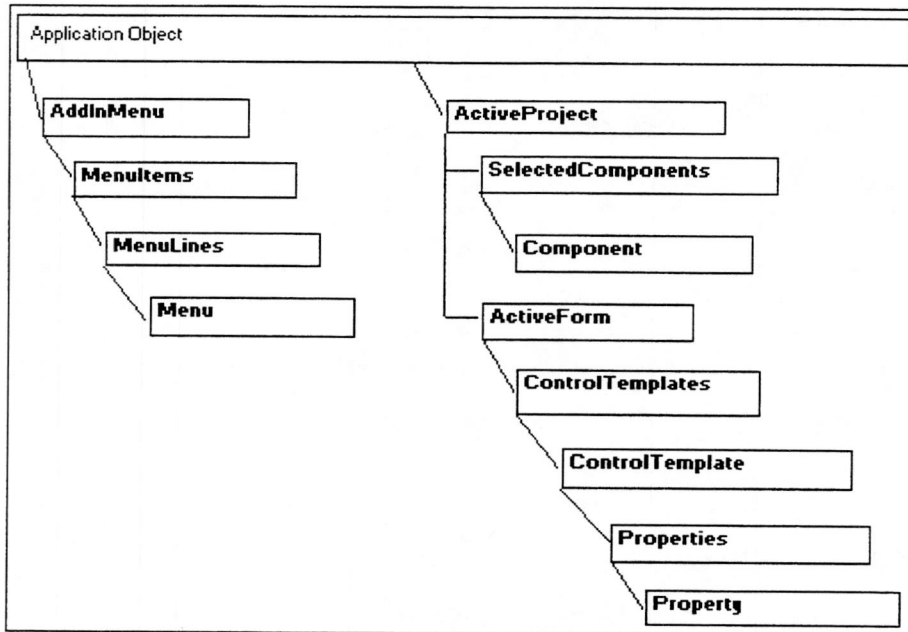


Figure 1-27: Add-In Objects.

```
' Store a handle to the Add-ins menu
Set pAppSubMenu = appVBInstance.AddInMenu.MenuItems

' Add a sub menu item under the Add-ins menu
Set pAppMyMenu = pAppSubMenu.Add("Command Button")

' Get connection handle and tell it to call this object
' when the Click event on that menu happens.
plngConnectID = pAppMyMenu.ConnectEvents(Me)
End Sub
```

Listing 1-27: ConnectAddIn() Event.

```
Sub DisconnectAddIn(ByVal intMode As Integer)
' Disconnect the event handler for our menu
pAppMyMenu.DisconnectEvents plngConnectID

' Remove the menu item from the Visual Basic instance
pAppSubMenu.Remove pAppMyMenu
End Sub
```

Listing 1-28: DisconnectAddIn() Event.

```
Public Sub AfterClick()
Dim ctlTemp As ControlTemplate

Set ctlTemp = pAppVBInst.ActiveProject._
ActiveForm.ControlTemplates.Add("CommandButton")
End Sub
```

Listing 1-29: AfterClick() Event.

The Sub Main routine should take care of adding the line in the Add-ins section of the VB.INI file:

```
Sub Main()
Dim strSection As String
Dim strString As String
Dim intLen As Integer

Const PROJ_NAME = "CommandAdd.AddCommand"

' Initialize section name based on OS
#If Win16 Then
strSection = "Add-Ins16"
#Else
strSection = "Add-Ins32"
#End If

' Initialize string
strString = Space$(100)

' See if this Add-in is already in the
' VB.INI file.
intLen = OSGetPrivateProfileString(strSection, PROJ_NAME, _
"Not Found", strString, Len(strString) + 1, "VB.INI")
If Left$(strString, intLen) = "Not Found" Then
intLen = OSWritePrivateProfileString(strSection, PROJ_NAME, _
"0", "VB.INI")
End If
End Sub
```

Listing 1-30: Sub Main() Routine.

What Is An Add-In?

The Visual Basic 4.0 design environment can be controlled with an Add-In via OLE Automation. In this section, you will see how it is done and see an example of an add-in that can help you upgrade Visual Basic 3.0 applications.

From the Add-ins menu you can activate a tool called the Add-in Manager. This tool shows you a list of the Add-ins available in your Visual Basic 4.0 environment. Add-ins have to be registered in your system registry, and they have to be listed in the [Add-ins32] or [Add-ins16] section in your VB.INI file.

You can create Add-ins using either Visual Basic 4.0, Visual C++ or any other tool capable of creating an OLE server. Here are the basic steps to creating an Add-in:

Create A New Project

1. Select Tools, References and make sure the "Microsoft Visual Basic 4.0 Development Environment" line is selected.
2. Insert a .BAS module.
3. Create a Sub Main procedure. In this routine add an entry under the Add-ins32 or Add-ins16 section in the VB.INI file.
4. Insert a Class Module.
5. Set the Instancing property to 1 or 2.
6. Set the Public property to True.
7. Set the Name to a name for the Add-in.
8. Select the Tools, Options, Project tab and set the Project Name and Application Description to names that identify this add-in. The Project Name is used in the Object Browser, and the Application Description is used in the Add-in manager.
9. From the Project tab, set the Start-up mode to OLE server.
10. Add the ConnectAddIn(), DisconnectAddIn(), and AfterClick() events to your class module. Refer to the listings below.
11. Create the following Private properties in your class:

```
Private pAppVBInst As VBIDE.Application
Private pAppSubMenu As VBIDE.MenuItems
Private pAppMyMenu As VBIDE.MenuLine
Private plngConnectID As Long
```

Listing 1-26: Private Properties Necessary For An Add-In.

```
Sub ConnectAddIn(appVBInstance As VBIDE.Application)
    ' Store the Visual Basic instance handle
    Set pAppVBInst = appVBInstance
```

```
' Put file name and extension together to check
' for valid characters.
pstrConvString = pstrConvString & strExt
intLen = Len(pstrConvString)
For intLoop = 1 To intLen
    strChar = UCase$(Mid$(pstrConvString, intLoop, 1))
    If (IsNumeric(strChar)) Or
        (InStr(ALPHA, strChar) > 0) Or
        (InStr(OTHER, strChar) > 0) Then
        ' Letter is OK
    Else
        FileValid = False
        Exit For
    End If
Next intLoop
End Function
Listing 1-25: FileValid() Function.
```

```
' Now go backwards to first "\"
intLen = Len(pstrConvString)
intLoop = intLen
Do Until Mid$(pstrConvString, intLoop, 1) = "\"
    intLoop = intLoop - 1
Loop
FileNameGet = Right$(pstrConvString, intLen - intLoop)
End If
End Function
Listing 1-23: FileNameGet() Function.
```

```
Function FileExtGet() As String
    Dim intPos As Integer

    ' get rid of any .. that may be in the path
    pstrConvString = StringTran("../", "")

    intPos = InStr(pstrConvString, ".")
    If intPos > 0 Then
        FileExtGet = Right$(pstrConvString, _
            Len(pstrConvString) - intPos)
    Else
        FileExtGet = ""
    End If
End Function
Listing 1-24: FileExtGet() Function.
```

```
Function FileValid() As Integer
    Dim intLoop As Integer
    Dim intLen As Integer
    Dim intPos As Integer
    Dim strChar As String
    Dim strPath As String
    Dim strExt As String

    Const ALPHA = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    Const OTHER = "_^$~!#%&-{}()@'`"

    ' Change the converted string
    pstrConvString = pstrString

    FileValid = True

    strPath = FilePathGet()           ' Get path
    strExt = FileExtGet()             ' Get extension
    pstrConvString = FileNameGet()    ' Now get file name

    ' Check length of file name
    If (Len(pstrConvString) > 8) Or (Len(pstrConvString) = 0) Then
        FileValid = False
    End If
    ' Check length of extension
    If Len(strExt) > 3 Then
        FileValid = False
    End If
```

other text boxes on the screen. Four different methods extract file information from the String class: `FilePathGet()`, `FileNameGet()`, `FileExtGet()`, and `FileValid()`. Each method is listed below:

```
Function FilePathGet() As String
    Dim intPos As Integer
    Dim intLen As Integer
    Dim intLoop As Integer

    ' Change the converted string
    pstrConvString = pstrString

    ' Strip out path
    intPos = InStr(pstrConvString, "\")
    If intPos > 0 Then
        intLen = Len(pstrConvString)
        For intLoop = intLen To 1 Step -1
            If Mid$(pstrConvString, intLoop, 1) = "\" Then
                Exit For
            End If
        Next intLoop
        FilePathGet = Left$(pstrConvString, intLoop)
    Else
        FilePathGet = ""
    End If
End Function
```

Listing 1-22: `FilePathGet()` Function.

```
Function FileNameGet() As String
    Dim intLen As Integer
    Dim intLoop As Integer
    Dim intPos As Integer

    ' Change the converted string
    pstrConvString = pstrString

    If Len(pstrConvString) = 0 Then
        ' No file name, no path
        FileNameGet = ""
    ElseIf Mid$(pstrConvString, Len(pstrConvString), 1) = "\" Then
        ' No file name, just a path
        FileNameGet = ""
    ElseIf InStr(pstrConvString, "\") = 0 Then
        ' No path just a file name
        ' Strip out extension
        intPos = InStr(pstrConvString, ".")
        If intPos > 0 Then
            FileNameGet = Left$(pstrConvString, intPos - 1)
        Else
            FileNameGet = pstrConvString
        End If
    Else
        ' Get rid of any .. that may be in the path
        pstrConvString = StringTran("../", "")

        ' Strip out extension first
        intPos = InStr(pstrConvString, ".")
        If intPos > 0 Then
            pstrConvString = Left$(pstrConvString, intPos - 1)
        End If
    End If
End Function
```

public interface can always stay the same. For example, if you wish to change the name of a private property in a Class, you can change it in one place, and you do not need to change the Public interface. You simply need to change the one Property Procedure to access this renamed property.

Type in the first line shown in the code below and press Enter. Then you can enter the rest of the code. This statement allows you to use the code shown in the sample above.

```
Property Let TheString(ByVal TheString As String)
    pstrString = TheString
End Property
Listing 1-19: TheString's LET Method.
```

When you set a property that is mapped to a Property Procedure; such as `oString.TheString = "This String Here"`, the Property Let procedure is fired. The value on the right hand side of the equal sign is given to the parameter in the Property Let procedure. You can then do whatever you want to that parameter before assigning it to the private property.

Property Get Procedure

To read the information from a private property in a Class, you need to create a Property Get procedure. This is exactly like a function in a .BAS module. It will return a value to the variable or expression on the left hand side of the equal sign, as in `strString = oString.TheString`:

```
Property Get TheString() As String
    TheString = pstrString
End Property
Listing 1-20: TheString GET Method.
```

Retrieving File Information

The second button on the sample form in the String Class that you created a few pages ago allows you to enter a file path and name, and it fills in the rest of the information on the form:

```
Private Sub cmdFile_Click()
    Dim oFile As New clsString

    If Trim$(txtFile.Text) <> "" Then
        oFile.TheString = txtFile.Text
        txtPath.Text = oFile.FilePathGet()
        txtFileName.Text = oFile.FileNameGet()
        txtExt.Text = oFile.FileExtGet()
        txtValid.Text = oFile.FileValid()
    End If
End Sub
Listing 1-21: cmdFile_Click() Event.
```

In Listing 1-21, you take the value from the Text Box named `txtFile` and assign it to the `TheString` property of the String class. Next you invoke different methods of the String class to fill in the

In Listing 1-17, you dim a variable as a New clsString and set the TheString property to a string with many spaces between the words. The property TheString is actually a Property Let method in the String Class module. Next you invoke the StringStrip() method of the string object passing a single space character. The output from the MsgBox looks like the following:

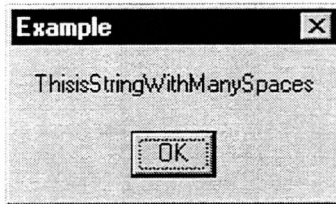


Figure 1-26: Results Of the StringStrip Method.

The StringStrip() method removes all occurrences of the character from the original string stored in the object. The following code is contained in the clsString Class:

```
Public Function StringStrip(ByVal strStrip As String) As String
    Dim intPos As Integer

    ' Get the original string
    pstrConvString = pstrString

    intPos = InStr(pstrConvString, strStrip)
    Do Until intPos = 0
        pstrConvString = Mid$(pstrConvString, 1, intPos - 1) + _
            Mid$(pstrConvString, intPos + 1)
        intPos = InStr(pstrConvString, strStrip)
    Loop

    ' Return the converted string
    StringStrip = pstrConvString
End Function
```

Listing 1-18: StringStrip() Method.

In Listing 1-18, you start by moving the original string to the pstrConvString property. You then do manipulations on the pstrConvString property. The users do not have to know anything about this property, since they retrieve everything from the TheString property or the methods.

Microsoft recommends NOT using Hungarian notation when naming properties. This helps users of your objects who may not necessarily be programmers. You may wish to name all private properties using Hungarian notation, and create Property methods for them. Users will be unable to manipulate those properties directly, and will be forced to use the Property procedures. This can help if you later wish to change the value that is returned to users: you can just change the Property procedure and do not have to rename variables within your Class.

Property Let Procedures

Instead of giving direct access to your properties within a class, you should create a Public Interface routine. Property procedures are used to both set and get properties within the classes you create. By using property procedures you can always change the underlying class and your

Class Modules

Visual Basic 4.0 can add classes to a project. Classes are used to create objects that you can manipulate in programs. There is no inheritance allowed between classes, but you should still find many uses for Classes in your projects. Here is a simple example to help you understand the idea of how to use classes. Later we will walk through a real-world example.

String Class

A class that can be very useful in many applications is a String class. This class can be responsible for stripping characters out of a string, extracting File/Path information, and many other things.

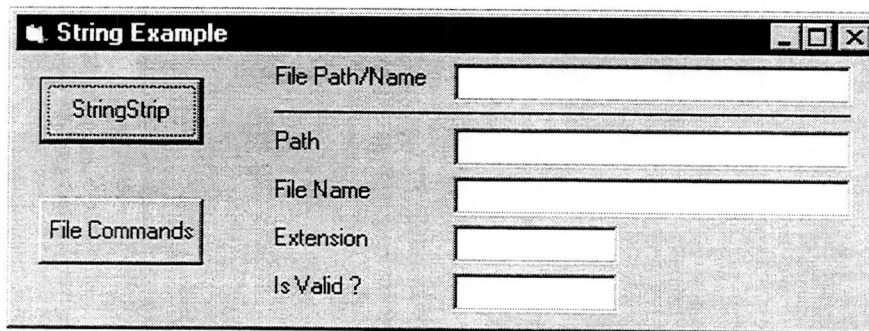


Figure 1-25: String Example.

Master Steps:

1. Add a new class by selecting **I**nsert, **C**lass Module from the main menu.
2. Press F4 and set the Name property to clsString.
3. Add two properties in the General Declarations area:

```
Option Explicit
Private pstrString As String
Private pstrConvString As String
```

The pstrString variable always contains the original string the user sets, while the pstrConvString is the converted string that the object manipulates. Under the StringStrip button, write the following code:

```
Private Sub cmdStrip_Click()
    Dim oStr As New clsString

    oStr.TheString = "This is String With Many Spaces"

    MsgBox oStr.StringStrip(" ")
End Sub
```

Listing 1-17: StringStrip Method.

Calling the Windows API

Making calls to the Windows API allows you to perform special actions that you can't perform directly with Visual Basic. In calling an API function, you must first specify for Visual Basic where the function is located and how it is defined. For example, if you want to make the computer produce the sound of the *default beep*, you will need to locate the proper API function which Windows uses to create beeps, in this case:

```
MessageBeep(0)
```

Listing 1-12: Windows API Function to Create Beeps.

Note The .WAV file that is played for the *default beep* is defined in your control panel under "Sounds".

Then you must determine which declare statement you need in order to tell Visual Basic where it is located and how it is called. An excellent tool to use is provided with Visual Basic called the API Text Viewer. Note that API calls are different for both 16-bit and 32-bit systems:

```
Declare Sub MessageBeep Lib "User" (ByVal N As Integer)
```

Listing 1-13: 16-Bit declare statement

```
Declare Sub MessageBeep Lib "User32" (ByVal N As Long)
```

Listing 1-14: 32-Bit declare statement

From this, it is easy to see why it is better to use as few API calls as possible—especially if you are developing for both 16- and 32-bit systems. Microsoft does not guarantee that API functions will migrate to future releases of Windows. Any API calls that you have in your application may fail in a future release of Windows.

For developers who develop applications that run in both 16- and 32-bit systems, Visual Basic provides you with conditional compiling. By using the #IF...THEN statements, you can instruct the compiler as to which block of code to use based on a Boolean value. For the MessageBeep function, you could use the following code:

```
#If Win32 Then
    Declare Sub MessageBeep Lib "User32" (ByVal N As Long)
#Else
    Declare Sub MessageBeep Lib "User" (ByVal N As Integer)
#End If
```

Listing 1-15: Declaring API Functions Using Conditional Compiling

Once the declare statement is defined, you can call the function just like any other Visual Basic function.

```
Sub Command1_Click()
    MessageBeep 0
End Sub
```

Listing 1-16: Sample Code Calling the Windows API Function "MessageBeep"

```
Private Sub cmdRead_Click()  
    dlgCommon.Filter = "RTF Files (*.rtf)|*.rtf"  
    dlgCommon.InitDir = "C:\"  
    dlgCommon.ShowOpen  
    If dlgCommon.filename = "" Then  
        MsgBox "No File Selected To Open"  
    Else  
        rtbNotes.LoadFile dlgCommon.filename, rtfRTF  
    End If  
End Sub
```

Listing 1-11: Reading an .RTF File.

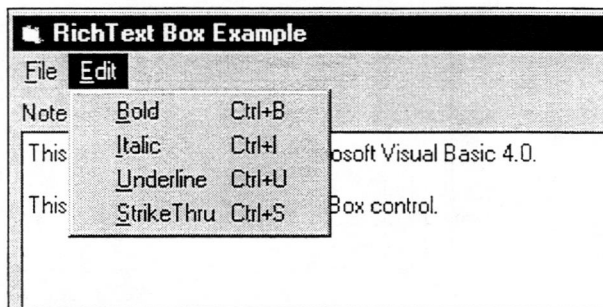


Figure 1-24: Edit Menu Of the Rich Text Box Example.

The text in the Rich Text Box changes after using one of the Edit menu items. You can also use the hot-keys associated with each menu item. The following code is placed behind each of the menu items:

```
Private Sub mnuEBold_Click()
    rtbNotes.SelBold = Not rtbNotes.SelBold
End Sub
Private Sub mnuEItalic_Click()
    rtbNotes.SelItalic = Not rtbNotes.SelItalic
End Sub
Private Sub mnuEUnder_Click()
    rtbNotes.SelUnderline = Not rtbNotes.SelUnderline
End Sub
Private Sub mnuEStrike_Click()
    rtbNotes.SelStrikethru = Not rtbNotes.SelStrikethru
End Sub
```

Listing 1-9: Selecting Text To Change The Style.

After changing the text, press the Save button to save the text in .RTF file format. This file can be read with Microsoft Word or any other word processor that reads .RTF files. To save the text in an .RTF format, use the following code:

```
Private Sub cmdSave_Click()
    dlgCommon.Filter = "RTF Files (*.rtf)|*.rtf"
    dlgCommon.InitDir = "C:\\"
    dlgCommon.ShowSave
    If dlgCommon.filename = "" Then
        MsgBox "File Not Saved"
    Else
        rtbNotes.SaveFile dlgCommon.filename, rtfRTF
    End If
End Sub
```

Listing 1-10: Saving an .RTF File.

Use the Common Dialog control to ask the user for the file name. Then invoke the SaveFile method of the RichTextBox control and give it the file name and file type. You are allowed to save it in an .RTF format or as simple text.

Or, for the sake of being complete with this example, here is how to read from an .RTF file using the LoadFile method:

false value. Try this sample to see how the message changes in the status bar as you move your mouse and cursor around the different text boxes.

RichTextBox

The RichTextBox control allows you to have a mini-word processor inside your Visual Basic application. This is useful for WYSIWYG editing within a text box. The RichTextBox control looks like the following in the toolbox:



Figure 1-22: RichTextBox Control.

Using properties of the RichTextBox control, you can change selected text to bold, italic, and strikethru, and you can change the selected text's color. You can adjust paragraph formatting by setting left and right indents, and you can create hanging indents.

Here is an example of the RichTextBox control and some of its features:

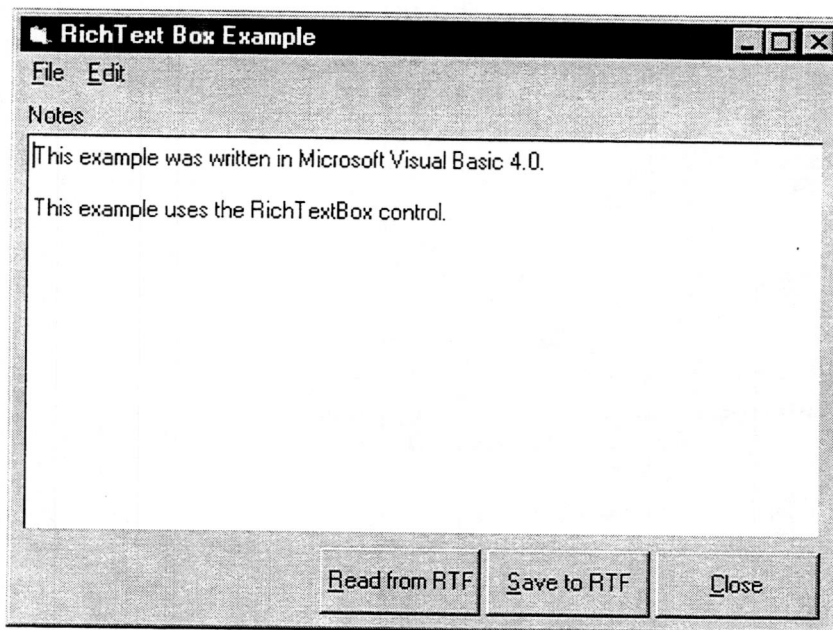


Figure 1-23: Rich Text Box Example.

Master Steps

1. Run the EXAMPLES.VBP project and select the button labeled "RichTextBox".
2. Highlight some text with the mouse.
3. Select one of the options from the Edit menu.

The GotFocus() event for each text box has a routine named StatusMsgShow(). This routine, responsible for displaying the text in Panel 1 of the Status Bar, is shown below:

```
Sub StatusMsgShow(ByVal strMsg As String)
    sbrStatus.Panels(1).Text = strMsg
End Sub
```

Listing 1-5: StatusMsgShow() Event.

In the above StatusMsgShow() procedure, the Panels collection references the number 1 panel that was defined to hold text information. The Text property was set with the strMsg variable that was passed into the routine.

When you tab from one control to the next, the message passed to the StatusMsgShow() routine displays. To make the message show up when you move the mouse over a particular text box, add the following code in the MouseMove() event for each text box:

```
Private Sub txtCompany_MouseMove(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    Call txtCompany_GotFocus
End Sub
```

Listing 1-6: txtCompany_MouseMove() Event.

In each MouseMove() event, the GotFocus() event was called, causing a flickering of the text as the mouse moved over the same text box. To avoid this, add the following code to the StatusMsgShow() procedure:

```
Sub StatusMsgShow(ByVal strMsg As String)
    If StatusDifferent(strMsg) Then
        sbrStatus.Panels(1).Text = strMsg
    End If
End Sub
```

Listing 1-7: StatusMsgShow() Event.

One of the problems with Listing 1-7 is that there is a flashing message inside the panel when you move the mouse continuously over the same text box. To resolve this, you need to determine if the same message is being passed to the StatusMsgShow() routine.

If the message being passed is the same as what is currently inside of Panel 1, do not refresh the text. If it is different, display the new text.

```
Function StatusDifferent(ByVal strMsg As String) As Integer
    If sbrStatus.Panels(1).Text = strMsg Then
        StatusDifferent = False
    Else
        StatusDifferent = True
    End If
End Function
```

Listing 1-8: StatusDifferent() Function.

The StatusDifferent() function looks at what is in the Text property of Panel 1 to see if the message text is different. If it is, the function returns a true value, otherwise the function returns a

Figure 1-20: Status Bar Example.

On the bottom of the form above is a status bar. It has a text area that contains the text "Company Name," and a date area (with the date). In Design Mode, if you click on the Custom property of the Status Bar control, you see the following dialog box:

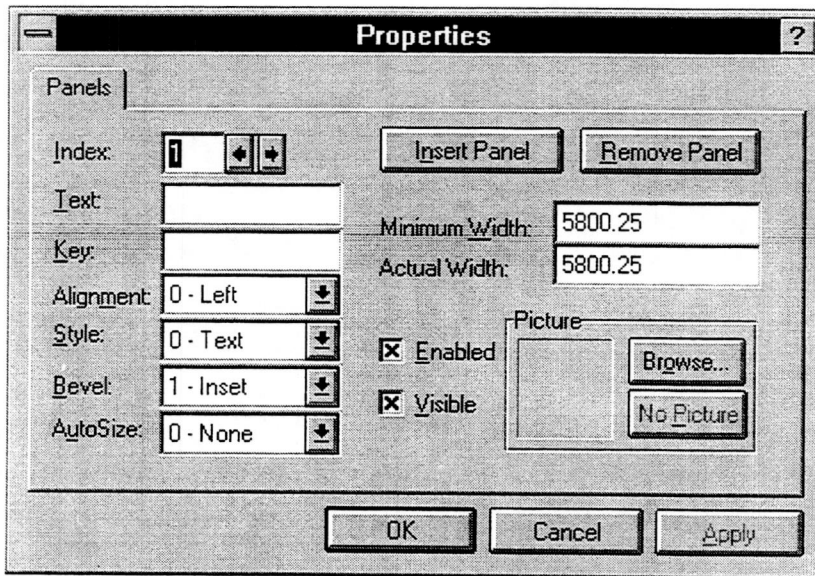


Figure 1-21: Panel Properties For Status Bar Control.

Each of the different status areas are Panel objects. The Status Bar control is made up of a Collection of Panel objects. Each panel object is defined using the Panel Properties dialog box shown in Figure 1-21.

The Index property is the number of the Panel property. Panels are numbered from 1 to n starting from the left hand side. You can set the alignment of the text within the panel using the Alignment property. You build panels from the left to the right. You cannot move the panels to the right hand side and build right to left. To determine what to put into each panel, you set the Style property.

On the sample, there are two panels. Panel 1 is set to a Text Style, Panel 2 is set to a Date Style. To display text in Panel 1 at runtime, write the following code in each Text Box on the form:

```
Private Sub txtCompany_GotFocus()  
    Call StatusMsgShow("Company Name")  
End Sub
```

Listing 1-3: txtCompany_GotFocus() Event.

```
Private Sub txtFirst_GotFocus()  
    Call StatusMsgShow("First Name")  
End Sub
```

Listing 1-4: txtFirst_GotFocus() Event.

Controls for 32-bit Visual Basic

There are additional controls in the 32-bit version of Visual Basic 4.0 that are not in the 16-bit version of Visual Basic 4.0, including:

- Status Bar
- Slider
- ImageList
- RichTextBox
- Toolbar
- ProgressBar
- TabStrip
- ListView

These controls are contained in a file called COMCTL32.OCX which you need to distribute to the user. If you are running in 16-bit Visual Basic, these controls will not appear in your Windows toolbox.

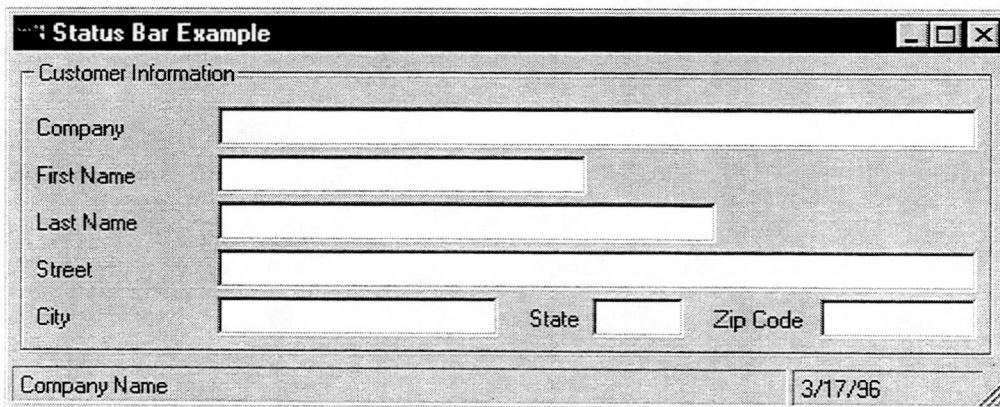
Two of the controls, Status Bar and RichTextBox, warrant discussion in detail.

Status Bar

The Status Bar control allows you to create a status area, like you see at the bottom of Microsoft Word or Microsoft Excel. You can display the Date, Time, Caps Lock, Num Lock, Scroll, Insert, or Text. The Status Bar control looks like the following in the toolbox:



Figure 1-19: Status Bar Control.



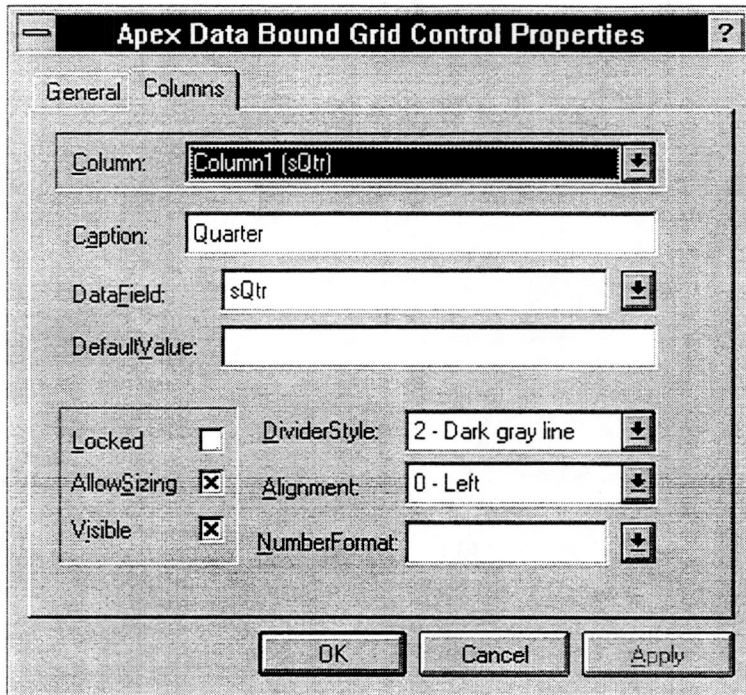


Figure 1-18: Apex Data Bound Grid Control Properties.

After changing to the Combo Box, you need to synchronize the Combo Box with the Customer Data control. To accomplish this, follow the Master Steps.

Master Steps

1. Set the DataSource property of the dbcState combo box to "datCust".
2. Set the DataField property to "sState_cd".

This links the value in the tblCustomers.sState_cd field to this combo box.

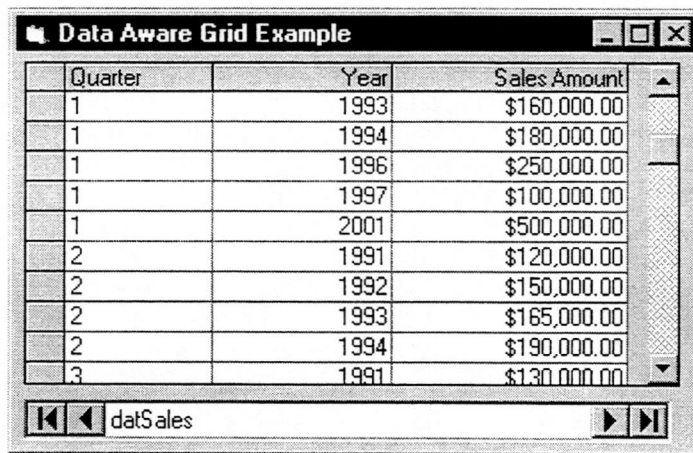
Data Aware Grid

The Data Aware Grid, from Apex Software, is very useful for displaying and editing data that is typically viewed in a spreadsheet format. Apex Software is the maker of the TrueGrid control. The Data Aware Grid control looks like the following in the toolbox:



Figure 1-16: Data Aware Grid.

You can create the following form, a table that contains sales figures for several years broken out by quarters with the Data Aware Grid:



Quarter	Year	Sales Amount
1	1993	\$160,000.00
1	1994	\$180,000.00
1	1996	\$250,000.00
1	1997	\$100,000.00
1	2001	\$500,000.00
2	1991	\$120,000.00
2	1992	\$150,000.00
2	1993	\$165,000.00
2	1994	\$190,000.00
3	1991	\$130,000.00

Figure 1-17: Data Aware Grid Example.

You can also change the Caption of each column in the grid by using the Apex Data Bound Grid control properties.

To synchronize the list box with the data aware control, add the following code in the Click event of the list box:

```
Private Sub dblCust_Click()  
    datCust.Recordset.Bookmark = dblCust.SelectedItem  
End Sub
```

Listing 1-2: Synchronize List Box And Data Control.

By taking the value of the SelectedItem property from the list box and assigning it to the Bookmark property of the data control, you force the data control to update the Recordset object.

The data aware controls are very robust. You could easily load over 6,000 names using a Dynaset object very quickly. These controls appear to buffer the information, read only a little data at a time, and then go back to the table when necessary.

Data Aware Combo Box

The Data Aware Combo Box is similar to the Data Aware List Box. It allows you to load a combo box from a data control with no coding on your part. The Data Aware Combo Box control looks like the following in the toolbox:



Figure 1-14: Data Aware Combo Box.

Using the Combo Box, you can change the State Text Box to a Combo Box.

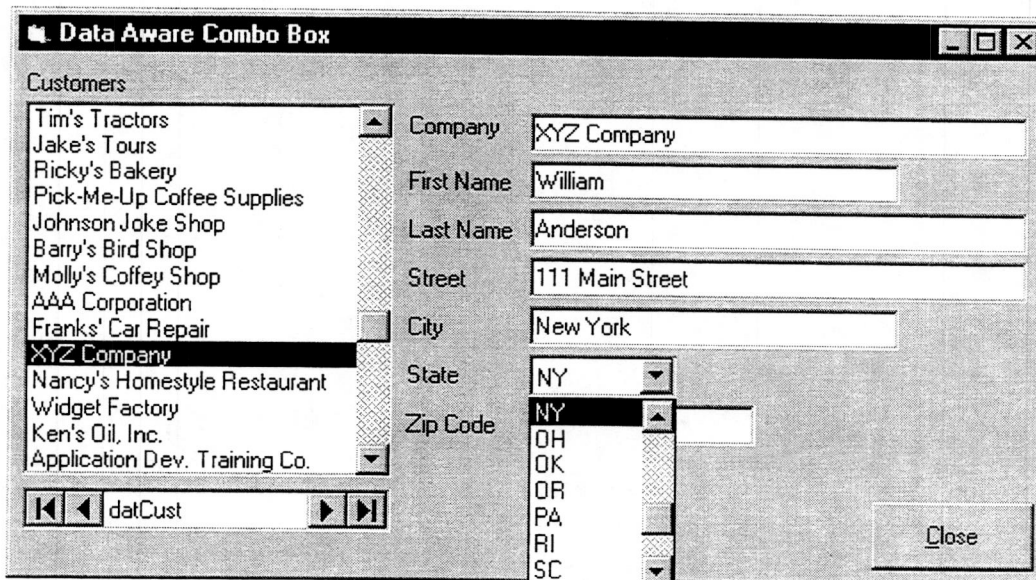


Figure 1-15: Data Aware Combo Box Example.

Figure 1-11: Data Aware List Box.

Using the Data Aware List Box, you can create a form that displays all company names in a list box.

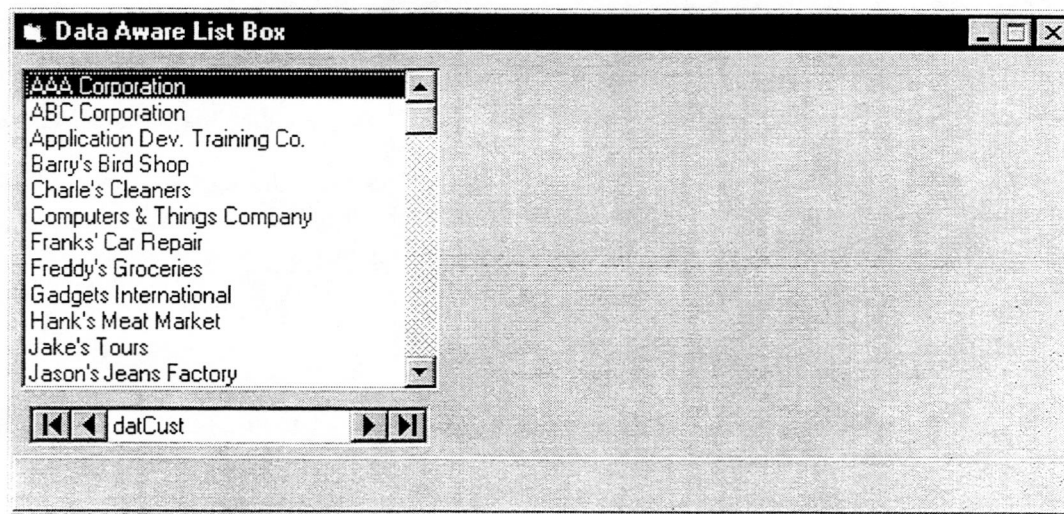


Figure 1-12: Data Aware List Box Control.

Using the Data Aware List Box, you can also update the data aware controls on that form by clicking on an item in the list box.

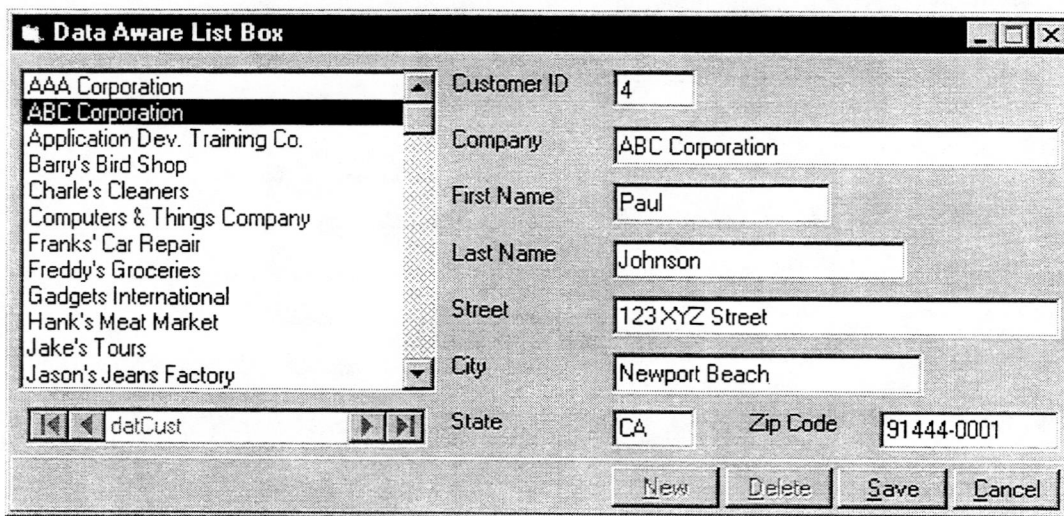


Figure 1-13: List Box Tied To Data Aware Controls.

You want the user to be able to click on an entry in the list box, and have the data control refresh the data aware controls on the right. To accomplish this, use the property called `SelectedItem` on the Data Aware List Box. This property contains the Bookmark of the row in the table that corresponds to the item in the list box.

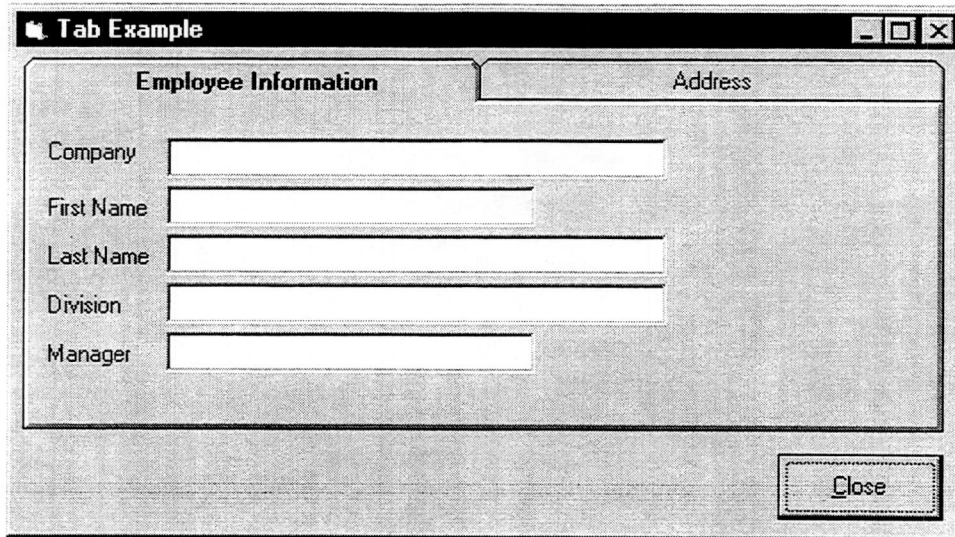


Figure 1-9: Sample Tab Control.

While this control does help organize information on one form, be careful about placing too many tabs on a form. The more controls you put on a single form, the longer that form takes to load.

You can set the Style property to either the Microsoft Office format (Figure 1-2), or the Windows 95 look. You may also align the tabs on the top, bottom, right or left of the Tab control.

TabStrip Control

The TabStrip control is one of the common controls in Windows 95 and Windows NT. It is very similar to the Tab Dialog control. The TabStrip control requires you to place a container control within each tab defined, and it allows you to assign tool-tips to each individual tab. Other than these two differences, the functionality is the same. The TabStrip control looks like the following in the toolbox:



Figure 1-10: TabStrip Control.

Data Aware List Box

You can use the Data Aware List Box whenever you wish to use a data control to automatically load information from a table into a list. This can be done with no coding on your part. The Data Aware List Box looks like the following in the toolbox:



Visual Basic 4.0 Custom Controls

There are many standard custom controls in both the 16 and 32-bit versions of Visual Basic. For those developers who may have used the previous release of Visual Basic, version 3.0, you may notice that some of your popular third-party custom controls have now been incorporated right into this new release.

ActiveX™ Controls

Both VBX controls and ActiveX™ Controls are two different forms of OLE controls. The practical difference between the two is that VBX controls are used only with the 16-bit version of Visual Basic. If you are migrating to Visual Basic 4.0 from a prior version, Microsoft recommends upgrading your VBX controls to ActiveX™ Controls. The advantages of moving to ActiveX™ Controls are:

- Methods are unique to each ActiveX™ Control
- Other applications, including Microsoft Access for Windows 95 and Microsoft Visual FoxPro, can use these controls
- Portability between 16-bit and 32-bit systems.

Tab Dialog Control

The tab dialog box actually comes from Sheridan Software Systems. Use the Tab Dialog control when you have more information than fits on one screen. The Tab Dialog control looks like the following in the toolbox:



Figure 1-8: Tab Dialog Control.

Listing 1-1: Command Button Click Event.

3. Save the code and run the application by clicking on the Run menu, and then press Start or the F5 key. Visual Basic will prompt you to save the form and the project. You may accept the default names. Your first running Visual Basic application looks like this:

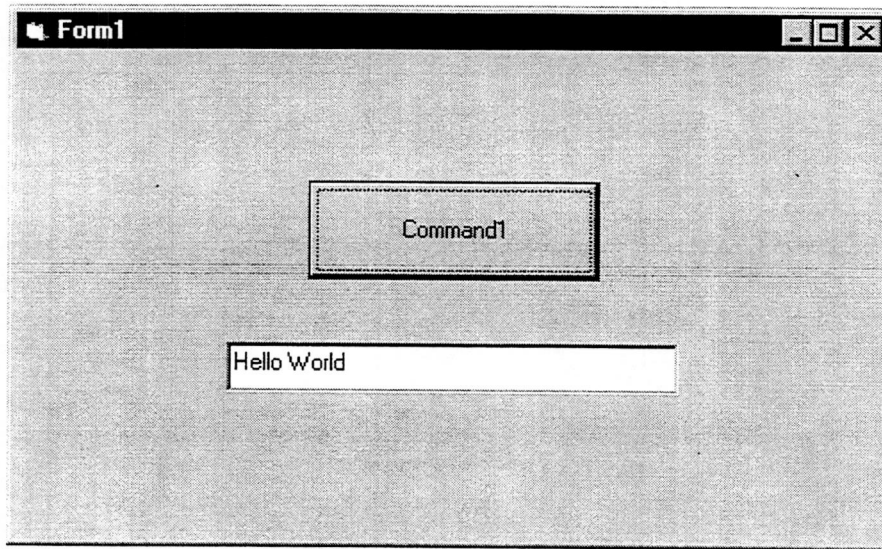


Figure 1-7: The Completed Application

4. Double-click on the close window box (the X in the top right corner of the form) to stop the program from running and return Visual Basic to design view.

Feel free to experiment with the standard control and familiarize yourself with how they operate. In the next section you will look at some of the more complex controls that are available to give your application a more professional look.

Master Steps

1. Double-click the mouse on the command button called Command1. Visual Basic takes you to the list of event procedures for that control.

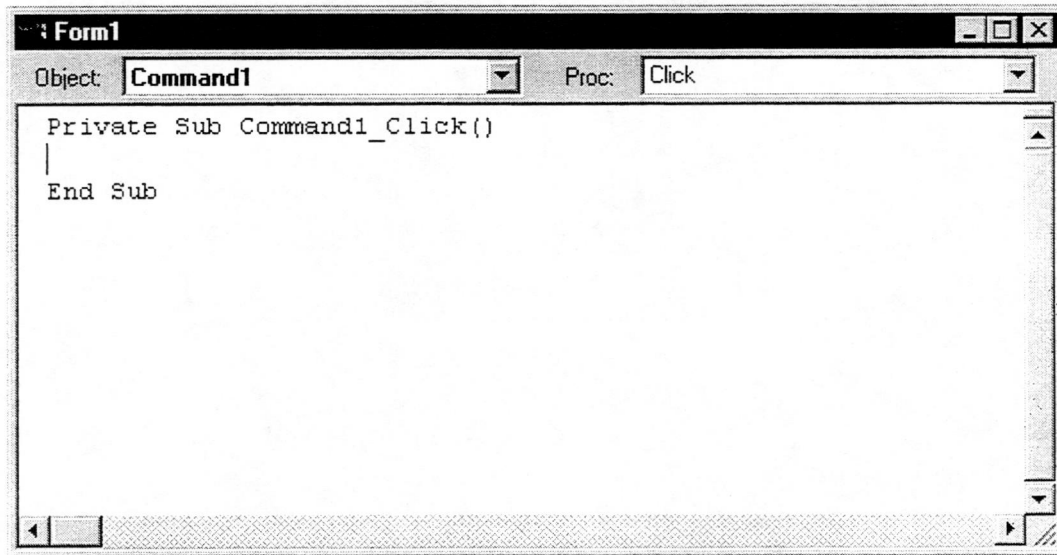


Figure 1-6: Coding Windows for the Command_Click Event.

Note Visual Basic defaults to the Click event procedure (event) for the command button control called Command1. If you were to scroll the list box of procedures, you'll see other events of this command button control that you could write code for. If you scroll the list box for objects, you'll see that you can write event procedures for the form itself (Form1) or any of the controls on that form (the command button control called Command1, and the text box control called Text1).

2. Click the mouse pointer so that it falls on the space between the beginning and end of the code template for the click procedure of Command1. Type in the missing comments (beginning with a single apostrophe) and single line of code until you have the following:

```
Private Sub Command1_Click()  
    ' The following code is executed each time a  
    ' click event occurs to the CommandButton labeled  
    ' "Command1".  
  
    ' This line changes the TextBox property  
    ' "Text" to contain the following string.  
    Text1.Text = "Hello World"  
  
End Sub
```

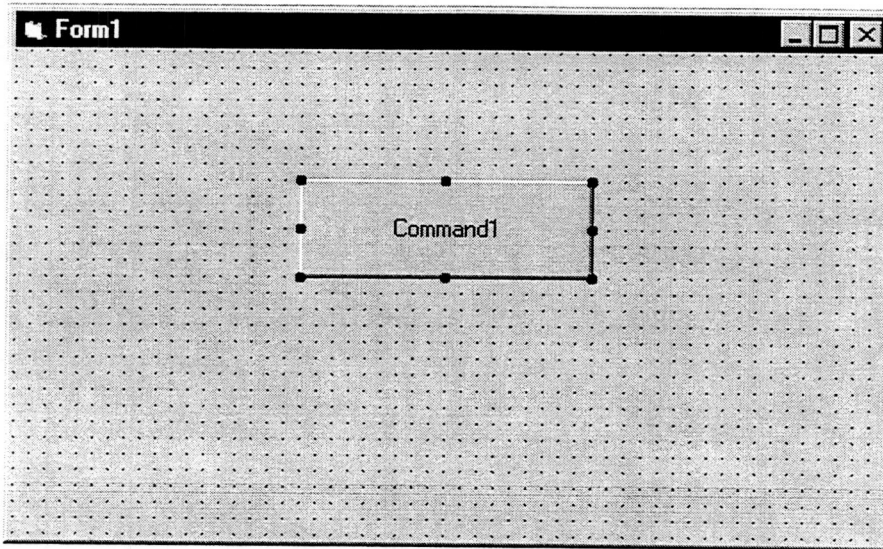



Figure 1-4: Form with Command Button Control.

Now add a text box control using this same technique.

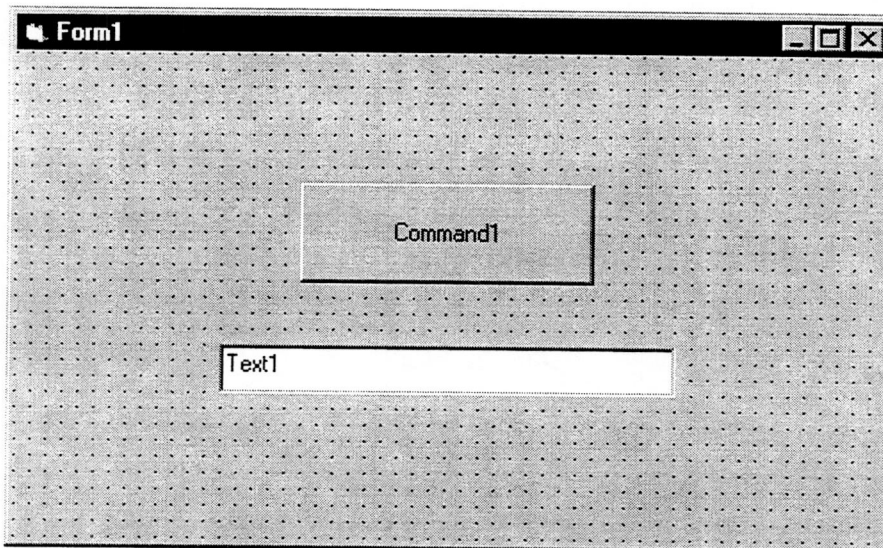


Figure 1-5: Form with Command Button and Text Box Controls.

Adding Events To Controls

Each control has its own set of events that get triggered each time the event occurs. For example, the click event gets triggered each time the user clicks on that control. To demonstrate this, follow the steps to add code to the Click event of the command button control (called Command1 from the previous example) to display the text “Hello World” within the text box control (called Text1 from the previous example).

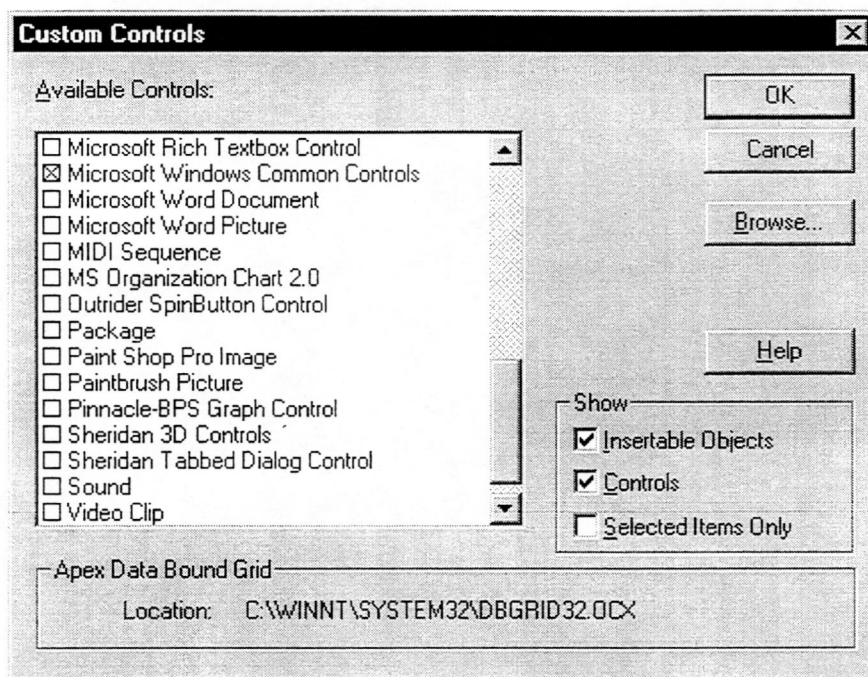


Figure 1-3: Add A Custom Control To The Default Toolbox.

You can use this dialog box to select and de-select controls from your Visual Basic Project.

Once you have used the dialog box above to indicate which controls you need for your application, placing a control on to a form is as easy as clicking and dragging. Using the mouse, click on the control you desire to place on the form. Then click the mouse on the form to designate the upper-left corner of where the control will be placed. Then drag the mouse to the lower-right corner. As an example, here is how to add a command button (push button) to a form.

Master Steps

1. Click on the command button control located in the Custom Control Toolbox.
2. With the mouse pointer anywhere on the form, Click the left mouse button down and hold it, visually dragging a sizing box on the form. Make the control as large as you want it and release the left mouse button.

Your form will look something like this:

Adding Controls To The Form

When you start a project, you will find the Custom Control Toolbox which looks like this:

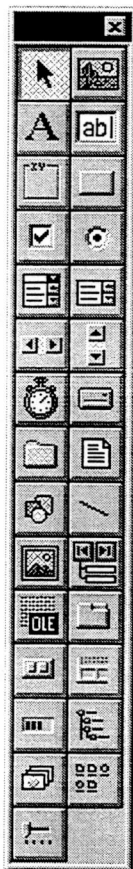


Figure 1-2: Custom Control Toolbox.

The above figure shows all the standard Visual Basic controls. To add additional controls to this toolbox, select the menu option Tools ... Custom Controls or use the hotkey CTRL-T. This will bring up the following dialog box:

Visual Basic 4.0 Basics

Most Visual Basic applications contain a few key components: Project File, Form Modules, Class Modules, Standard Modules, and ActiveX™ Controls. This section will take a look at each component and how it fits into your application.

Project Files

Project files have the extension .VBP and contain references to all Form Modules, Class Modules, Standard Modules, and ActiveX™ Controls that are used in your Visual Basic application. To create a new project, select **FILE - New Project** from the Visual Basic File menu. To add and remove modules from your project, use the **Add File** and **Remove File** menu options. When you have completed all the modules in your project, you can then use the **Make EXE File** or **Make OLE DLL** menu option to create an executable or an OLE Dynamic Link Library of the project.

Form Modules

Forms (.FRM filename extension) are the foundation of any Visual Basic application. A .FRM file contains descriptions of the form, description of all controls placed on the form, and all code associated with the form. After you create a new project, Visual Basic creates a default form called FORM1. It will look like the following:

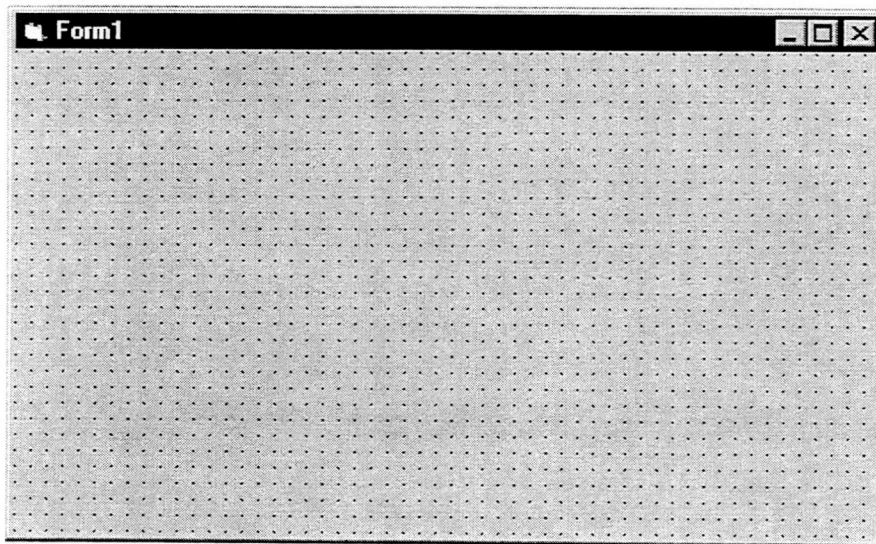


Figure 1-1: Form1 In A New Visual Basic Project.

You have access to many properties associated with Form1. For example, some of the properties that effect the appearance of the form include: background color, foreground color, border style, caption, font, height and width. Other properties that you can modify include the name of the form, its screen location, and what default icon is displayed when the form is minimized.

- * Resource files can be used to create multi-lingual versions of your applications

- **Jet Engine**
 - * Jet 2.5 (16-bit) and Jet 3.0 (32-bit) engines with full database object programmability
 - * Full database creation from Visual Basic code including referential integrity
 - * Replication support with Jet 3.0
 - * Data-aware controls
 - * List Box, Combo Box, and Grid
- **32-bit Custom Controls**
 - * Status Bar
 - * Slider
 - * Image List
 - * List View
 - * Tab Strip
 - * Tool Bar
 - * Rich Text Box
 - * Progress Bar
- **Add-ins**
 - * Add-ins menu allows third-party vendors to add functionality to the IDE
 - * Can write Add-ins in Visual Basic
 - * Full object hierarchy exposed in IDE
 - * Control Add-ins via OLE Automation
- **Enhanced OLE Support**
 - * Support for in-place activation
 - * Named arguments make OLE Automation easier to code
- **OLE Automation Servers**
 - * Ability to create OLE servers in Visual Basic
 - * Can control these servers from any OLE application
 - * Remote OLE servers
 - * Ability to run OLE server on another machine on a network
- **Other Features**
 - * Setup wizard
 - * Network setup and uninstall
 - * Incremental compiling
 - * Demand compiling
 - * Project loading in the background

Visual Basic 4.0 Features List

- **Visual Basic for Applications (VBA)**
 - * Visual Basic For Applications is the new language
 - * Compatible with Access 95, Excel 5.0, Excel 7.0, Project 4.0, and Project 7.0
 - * Conditional Compilation
 - * Allows you to target 16-bit and 32-bit with one set of source code
 - * Built-in constants
 - * Only those constants used are compiled into the final .EXE file
 - * More data types: Boolean, Date, and Byte
- **ActiveX™ Controls, formerly called OLE controls**
 - * Support for 16-bit VBXs and 16-Bit ActiveX™ Controls in Windows 3.x
 - * 32-bit ActiveX™ Controls in Windows 95 and Windows NT
 - * 16-bit VBX support in Windows 95 and Windows NT via *Windows On Windows* (WOW)
 - * Automatic morphing from VBX controls to ActiveX™ Controls
 - * Over 140 third-party vendors are updating their VBXs to ActiveX™ Controls
 - * Reusable component can be used in Visual Basic, Microsoft Access, Visual FoxPro and Visual C++
- **Visual Basic Integrated Development Environment (VBIDE)**
 - * Tool-tips
 - * Right-mouse button is now enabled
 - * Ability to lock controls
 - * Options to allow you to customize your environment
 - * Can be controlled via OLE Automation for Add-ins capability
- **Class Modules**
 - * Ability to create Classes
 - * Great for creating reusable components
 - * Polymorphism
 - * Encapsulation
 - * Objects can be encapsulated within Objects
 - * Custom Properties
 - * Property Procedures

The Remote Data Objects (RDO) and Remote Data Control are new tools that allow you to communicate via ODBC directly instead of using Jet as a layer on top of ODBC. With RDO, you still use data aware controls. A Component Manager allows you to manage reusable code created by several programmers. This should help prevent large teams from reinventing the wheel.

The Enterprise Edition can be used by any developer wanting to do true client/server applications using a professional database server such as Microsoft SQL Server 6.x or Oracle 7.x.

The extra cost of the Enterprise Edition is well worth the documentation and additional tools you receive.

Editions Of Visual Basic 4.0

Visual Basic 4.0 comes in three editions:

- Standard Edition
- Professional Edition
- Enterprise Edition

Features Of The Standard Edition

Visual Basic 4.0 Standard Edition supports only 32-bit development platforms, such as Windows 95 and Windows NT 3.51 or later. You cannot run the standard edition in Windows 3.x. Visual Basic 4.0 Standard Edition has all of the standard Windows controls, plus data aware controls. None of the programmable Data Access Objects (DAO) are available in the standard edition, however, including the Database and Recordset objects. For documentation, you receive a printed *Programmer's Guide*; the rest of the help is online.

Visual Basic 4.0 Standard Edition is an introductory tool to Visual Basic for simple programming and hobbyists. Serious developers will prefer Visual Basic 4.0 Professional Edition or the new Visual Basic 4.0 Enterprise Edition.

Features Of The Professional Edition

Visual Basic 4.0 Professional Edition supports both 16 and 32-bit development platforms. It includes all the features found in the standard edition, plus many more custom controls. Also included is the Crystal Report Writer Version 3.0 for creating reports for your applications. You can use all of the Data Access Objects (DAOs) from code, and you can create OLE servers. Documentation for this version includes the *Language Reference, Professional Features* book (describing all the custom controls), and the Crystal Reports manual. In addition to online help, you also receive Visual Basic Books Online, the printed documentation in an online format.

Visual Basic 4.0 Professional Edition is for developers doing development against desktop databases such as Microsoft Access, Xbase or other file-server, ISAM-type files.

Features Of The Enterprise Edition

The Enterprise Edition is for the serious corporate developer, Visual Basic development teams, or the developer who creates sophisticated Windows applications. The documentation that comes with this version includes a great tutorial on client/server development and ODBC connectivity. This version includes information on building OLE servers in Visual Basic and Remote OLE servers. You also receive a copy of Microsoft SourceSafe, a version control system.

Platforms Supported

Visual Basic 4.0 supports both 16 and 32-bit operating environments/systems for any one of the following Windows platforms:

- Windows 95 (32-bit)
- Windows 95 (16-bit supported with *Windows On Windows* (WOW))
- Windows NT 3.51 or later (32-bit)
- Windows NT 3.51 or later (16-bit supported with WOW)
- Windows 3.x (16-bit)
- Windows for Workgroups 3.x (16-bit)

Microsoft is encouraging developers to migrate their development efforts towards the 32-bit environment. Windows 95 will be the platform for most users, although developers at large corporations may find Windows NT to be a more desirable environment. With the release of Windows NT 4.0, Windows NT will have the same GUI as Windows 95, giving it the same look and feel.

Any code you develop for Windows NT can run with little or no change on Windows 95, and vice versa. A few differences exist in the Windows API calls between Windows NT and Windows 95.

Introduction

Visual Basic 4.0, the latest from Microsoft in the line of Visual Basic products, provides a great deal of functionality and flexibility that you will want to add to your Visual Basic applications. In this chapter, we will take a quick look at the new language features, and in later chapters, we will talk about the features in more detail.

Objectives

In this section you will learn:

- Which platforms the different versions of Visual Basic 4.0 support?
- What are some of the features in Visual Basic 4.0?
- How do I create a Project?
- How do you use the Custom Controls?
- How do you access the Windows API
- What are all of the ways you can access databases from Visual Basic 4.0?
- How do you use Class Modules?
- What is available in Visual Basic 4.0 to help with version control?

Microsoft, Windows, and Win32, are registered trademarks and Visual Basic is a trademark of Microsoft Corporation. All other trademarks, marked and not marked, are property of their respective owners.

This document is provided for informational purposes only. The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to change in market conditions, it should not be interpreted to be a commitment on the part of Microsoft and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

INFORMATION PROVIDED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND FREEDOM FROM INFRINGEMENT. The user assumes the entire risk as to the accuracy and the use of this document. This document may be copied and distributed subject to the following conditions: 1) All text must be copied without modification (except foreign language translation) and all pages must be included; 2) All copies must contain Microsoft's and Application Developer Training Company's copyright notice and any other notices provided therein; and 3) **This document may not be distributed within the boundaries of Canada or the United States of America.**

Copyright © 1996 Application Developers Training Company
All Rights Reserved.